Date: February 2012

OBJECT MANAGEMENT GROUP

DDS for Lightweight CCM

Version 1.1 with change bars

OMG Document Number:	formal/2012-02-02
Standard document URL:	http://www.omg.org/spec/dds4ccm/
Associated File(s)*:	http://www.omg.org/spec/dds4ccm/20110201
	http://www.omg.org/spec/dds4ccm/20110201/ccm_dirl.idl
	http://www.omg.org/spec/dds4ccm/20110201/ccm_dds.idl
	http://www.omg.org/spec/dds4ccm/20110201/DDS_DefaultQoS.xml
	http://www.omg.org/spec/dds4ccm/20110201/DDS_QoSProfile.xsd

* original files: ptc/11-02-07, ptc/11-02-08, ptc/11-02-09, ptc/11-02-10

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

LICENSES

PATENTS

GENERAL USE RESTRICTIONS

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 140 Kendrick Street, Needham, MA 02494, U.S.A.

TRADEMARKS

MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™ , Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™ , IMM™, MOF™, OMG Interface Definition Language (IDL)™, and OMG Systems Modeling Language (OMG SysML)™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG’s Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents, Report a Bug/Issue (http://www.omg.org/technology/agreement.htm).

Table of Contents

Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at http://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling, and vertical domain frameworks. A catalog of all OMG Specifications is available from the OMG website at: http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

Business Modeling Specifications

Business Strategy, Business Rules and Business Process Management specifications

Middleware Specifications

- CORBA/IIOP
- Minimum CORBA
- CORBA Component Model (CCM)
- Data Distribution Service (DDS)

Specialized CORBA Specifications

Includes CORBA/e and Realtime and Embedded Systems

Language Mappings

- IDL / Language Mapping
- Other Language Mapping specifications

Modeling and Metadata Specifications

- UML®, MOF, XMI, and CWM specifications
- UML Profiles

Modernization Specifications

KDM

Platform Independent Model (PIM), Platform Specific Model (PSM), and Interface Specifications

- OMG Domain
- CORBAServices
- CORBAFacilities
- OMG Embedded Intelligence
- OMG Security

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. (as of January 16, 2006) at:

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: pubs@omg.org

Note – Certain OMG specifications are also available as ISO standards. Please consult http://www.iso.org

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to http://www.omg.org/technology/agreement.htm.

1 Scope

This specification defines how CCM[1] components may interact using DDS and how related DDS entities may be configured using CCM configuration mechanisms.

For that purpose, it uses the Generic Interaction Support recently added to CCM to allow extending CCM with new interactions. This support is made of two constructs: 1) a new port type (namely *extended port*) to capture as a whole a set of basic interactions that need to be kept consistent (a trivial example is e.g., how to provide message passing with flow control) and 2) abstractions in between components (namely *connectors*) to support new interaction mechanisms.

This specification thus defines DDS-dedicated extended ports and connectors. It is made of two parts.

- Chapter 7 defines extended ports and connectors for DDS-CDPS
- Chapter 8 defines extended ports and connectors for DDS-DLRL

This specification assumes an *a-priori* knowledge of the Generic Interaction Support. If not the case, refer to the CCM documentation.

2 Conformance

The conformance criteria of an implementation with respect to this specification is stated through the support for the following extensions:

1. ***A CCM framework claiming conformance with this "DDS for Lightweight CCM" specification*** shall support DDS-DCPS normative ports and connectors and their configuration.
2. ***An optional compliance point for this "DDS for Lightweight CCM"*** specification is the support for DLRL ports and connectors and their configuration.

3 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

- [**CORBA**] Common Object Request Broker Architecture: Core Specification, OMG, V3.2, part 1, part 2, and part 3 (formal/2011-11-01, formal/2011-11-02, formal/2011-11-03).
- [**CCM**] CORBA Component Model Specification refers to part 3 of the above-mentioned specification.
- [**D&C**] Deployment and Configuration of Component-based Distributed Applications, OMG, V4.0 (formal/06-04-02).
- [**DDS**] Data Distribution Service for Real-time Systems Specification, OMG, V1.2, (formal/07-07-01).

[1] In this document, CCM implicitly refers also to LightWeight CCM.

- [**XMLSchema**] XML Schema,W3C Recommendation, 28 October 2004. Latest version at
  http://www.w3.org/TR/xmlschema-1/
  and
  http://www.w3.org/TR/xml-schema-2/.

4 Terms and Definitions

In the scope of this specification, the following terms and definitions apply.

- **Connector** – Interaction entity between components. A connector is seen at design level as a connection between components and is composed of several fragments (artifacts) at execution level, to realize the interaction.
- **Extended Port** – Consists of zero or more provided as well as zero or more required interfaces, i.e., closely resembling the UML2 specification of a port.
- **Fragment** – Artifact, part of the connector implementation. A fragment corresponds to one executor that can be deployed onto an execution node, co-localized with one component for which it supports the interaction provided by the connector.

5 Symbols (and abbreviated terms)

The followings acronyms are intensely used in the following specification:

• CCM	CORBA Component Model
• CIF	Component Implementation Framework
• CORBA	Common Object Request Broker Architecture
• DCPS	Data-Centric Publish-Subscribe (part of DDS)
• DDS	Data Distribution Service
• DLRL	Data Local Reconstruction Layer (part of DDS)
• IDL	Interface Definition Language
• UML	Unified Modeling Language
• XML	eXtensible Mark-up Language

6 Additional Information

6.1 Changes to Adopted OMG Specifications

None in this specification.

6.2 Acknowledgements

The following companies submitted this specification:

- Mercury Computer Systems, Inc.
- PrismTech Group Ltd
- Real-Time Innovations, Inc.
- Thales

The following company supported this specification:

- Commissariat à l'Energie Atomique (CEA)

7 DDS-DCPS Extended Ports and Connectors

This chapter instantiates the Generic Interaction Support of CCM, in order to define ports and connectors for DDS-DCPS. This chapter assumes an *a-priori* knowledge of this CCM extension and of DDS specification, at least of its DCPS part.

7.1 Introduction

7.1.1 Rationale for DDS Extended Ports and Connectors Definition

DDS is a very versatile middleware. It allows accommodating almost any conceivable flavor of data-centric publish/ subscribe communication and therefore presents a very rich API and a very complete set of underlying behaviors and QoS policies. The counterpart of this richness is a certain complexity that may lead to errors or malfunctions due to mistaken uses.

Therefore, the purpose of "DDS for lightweight CCM" should be twofold:

1. Easing the deployment of applications made of components interacting through DDS by placing DDS configuration in the general component scheme (where configuration is carefully kept separated from the pure application code).
2. Providing to the components'author an easier access to DDS, by defining ready-to-use ports that would hide as much as possible DDS complexity.

However, ease of use should not come with too many restrictions that would compromise usefulness. In addition, as DDS is very versatile, defining a single couple of write and read ports that could accommodate simply all potential DDS usages seems unrealistic.

The process used to identify relevant DDS ports and connectors has been as follows:

- A large variety of DDS use patterns have been analyzed, then for each pattern, the roles1 have been identified and characterized in terms of:
  - associated DDS entities,
  - related QoS settings, and
  - programming contracts.
- All the identified programming contracts have been then analyzed and grouped to define DDS ports (each resulting programming contract corresponds to one DDS port).
- The most common DDS use patterns have been then identified as connectors, with their related DDS ports, their underlying DDS entities and associated QoS settings.

Even if these principles are general enough to be applicable to DCPS and DLRL uses of DDS, their actual realization results in extended ports and connectors that are specific to DCPS or DLRL.

7.1.2 From Connector-Oriented Modeling to Connectionless Deployment

It should be well understood that, even if at modeling levels DDS-enabled components are said ‘connected’ to a DDS-connector through their DDS-ports, that does not mean at all that they are physically connected (DDS is connectionless by nature). The following picture illustrates this change of paradigm from components connected to a DDS pattern at modeling time (in green) to components interacting via DDS through DDS ports to fulfill this DDS pattern at execution time (in yellow).

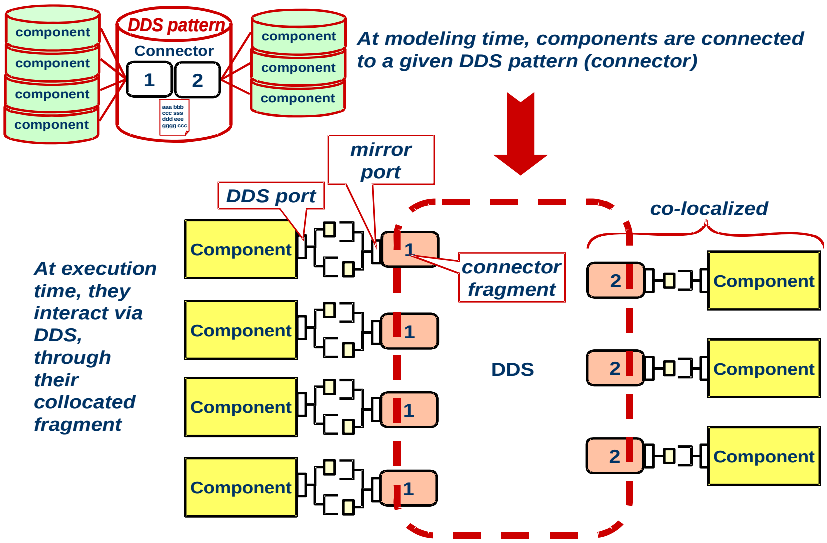


Figure 7.1 - From Modeling to Actual Deployment

7.2 DDS-DCPS Extended Ports

7.2.1 Design Rules

7.2.1.1 Parameterization

DDS-DCPS ports and connectors will be grouped in a module, itself parameterized by the data type and a sequence type of that data type.

- Grouping the definitions for port types and connectors in the same module allows that they share the same concrete interface when eventually instantiated.
- Passing that second parameter may seem redundant but it is the only way to allow sharing the sequence definition with the rest of the application.

To avoid useless duplications when instantiated, this template module will only contain the constructs that depend on the data-type. It will be included in a more general module that will also contain all the constructs that do not depend on the data-type.

Note – The following ports selected to be normative as fitting most DDS use patterns, are all parameterized by only one data type. However, as the Generic Interaction support allows defining new port types, nothing prevents users to define more specific ports that would be parameterized by several data types.

7.2.1.2 Basic Ports Definition

DDS-DCPS ports, as extended ports, will be made of several basic ports (**uses** and/or **provides**) with their defined interfaces.

The rationale to group operations as a single interface (thus one basic port), or on the contrary, to split them in different interfaces (thus several basic ports) is as follows:

- Different interaction directions (i.e., whether the component is a caller or a callee) result in different interfaces.
- Each interface is focused on a precise area of functionality (such as data access, status access...).

All those interfaces could be then considered as building blocks for DDS-DCPS extended ports.

7.2.1.3 Interface Design

For simplicity reasons, it has been chosen not only to keep the strictly needed operations, but also to simplify their parameters as much as possible, in particular:

- Information that comes with the read data samples have been simplified to what is most commonly used.
- Data access parameters, when they are likely to be shared by all the access of a given port (e.g., a query for read) are expressed by means of basic port interface attributes. Those attributes can be seen configurations for the ports.

Errors are reported by means of exceptions.

Sequences to be returned (of data and of accompanying information) are designed as ‘inout’ parameters, even if the actual information flow is only ‘out.’ This disposal allows for implementation of smarter memory management.

7.2.1.4 Simplicity versus Richness Trade-off

The goal of this specification is not to prevent the advanced user from making use of advanced DDS features, if needed. In return, complicating the mainstream port interfaces should be avoided. This is the reason why each DDS port contains an extra basic port to access directly to the more scoped underlying DDS entity (e.g., the **DataWriter** if it is a port for writing). If needed, all the involved DDS entities can be retrieved by this starting point.

Note: The DDS-DCPS ports are of large potential usage; however, as the Generic Interaction support allows defining new port types, nothing prevents users from defining their own DDS ports to fulfill more specific use patterns.

7.2.2 Normative DDS-DCPS Ports

This section lists the normative DDS extended ports. It starts with the list of interfaces for basic ports and then assembles them to make the DDS ports. All those constructs are included in the **Typed** template sub-module of the **CCM_DDS** module, as follows:

```
module CCM_DDS {
    // Non-typed definitions
    ...
    module Typed <typename T, sequence<T> TSeq> {
        // Typed definitions
        ...
        };
    };
```

The following sections list extracts from the template module **CCM_DDS::Typed<typename T, sequence<T> Tseq>**.

The whole consolidated IDL is listed in Annex A: IDL3+ of DDS-DCPS Ports and Connectors.

This IDL file is named "**ccm_dds.idl**."

7.2.2.1 DDS-DCPS Basic Port Interfaces

7.2.2.1.1 Data Access - Publishing Side

Two interfaces allow writing DDS data.

1. A **Writer** allows publication of data on a given topic without paying any attention to the instance lifecycle; therefore, it just allows writing values of the related data type.
2. An **Updater** allows publication of data on a given topic when you do care about instance lifecycle. It allows creating, updating, and deleting instances of the related data type. It can be configured to actually check the lifecycle globally or just locally.

The following IDL declarations of those related interfaces are followed by explanations when needed.

InstanceHandleManager

```
local interface InstanceHandleManager {
    DDS::InstanceHandle_t register_instance (in T datum)
        raises (InternalError);
    void unregister_instance (in T datum, in DDS::InstanceHandle_t instance_handle)
        raises (InternalError);
    };
```

This abstract interface gathers the two operations that allow manipulating DDS instance handles and will serve as a basis for the **Writer** or the **Updater** interfaces.

- **register_instance** asks DDS to register an instance, which results in allocating it a local instance handle. The targeted instance is indicated by the key value in the passed data (**datum**).
- **unregister_instance** asks DDS to unregister the instance, indicated by the passed **instance_handle** and the key values of the passed data (**datum**) and thus to release the instance handle.

Both operations are very similar to the DDS operations and are just passed to the DDS **DataReader** in support for the relater DDS port (see the DDS documentation for more details). Any DDS error will be reported through an **InternalError** exception.

Interface Writer

```
local interface Writer : InstanceHandleManager {
    void write_one (in T datum, in DDS::InstanceHandle_t instance_handle)
        raises (InternalError);
    void write_many (in TSeq data)
        raises (InternalError);
    attribute boolean is_coherent_write;// FALSE by default
    };
```

Behavior of a **Writer** is as follows:

- **write_one** allows publishing one instance value. The targeted instance is designated by the passed instance handle (**instance_handle**) if not **DDS::HANDLE_NIL** or else by the key values in the passed data (**datum**). If a valid handle is passed, it must be in accordance with the key values of the passed data; otherwise, an **InternalError** exception is raised with the returned DDS error code. More generally, any DDS error when publishing the data will be reported by an **InternalError** exception.
- **write_many** allows publishing a batch of instance values is a single operation. Resulting DDS orders are stopped at the first error (and the **index** of the erroneous instance value is reported in the raised **InternalError** exception). If the attribute **is_coherent_write** is **TRUE**, the resulting successful write DDS orders are placed between a DDS **begin_coherent_updates** and an **end_coherent_updates**.

Interface Updater

```
local interface Updater : InstanceHandleManager {
    void create_one (in T datum)
        raises (AlreadyCreated,
            InternalError);
    void update_one (in T datum, in DDS::InstanceHandle_t instance_handle)
        raises (NonExistent,
            InternalError);
    void delete_one (in T datum, in DDS::InstanceHandle_t instance_handle)
        raises (NonExistent,
            InternalError);

    void create_many (in TSeq data)
        raises (AlreadyCreated,
            InternalError);
    void update_many (in TSeq data)
        raises (NonExistent,
            InternalError);
    void delete_many (in TSeq data)
        raises (NonExistent,
            InternalError);

    readonly attribute boolean is_global_scope;    // FALSE by default
    attribute boolean is_coherent_write;           // FALSE by default
    };
```

Behavior of an **Updater** is as follows:

- **create_one** (resp. **update_one**, **delete_one**) allows creating (resp. updating, deleting) one instance. For **create_one** this instance is designated by the key value in **datum**. For the two others, it is designated by the passed instance handle (**instance_handle**) if not **DDS::HANDLE_NIL** or else by the key value in the passed instance data (**datum**). If a valid handle is passed, it must be in accordance with the key value of the passed instance data; otherwise, an **InternalError** exception is raised with the returned DDS error code. More generally, any DDS error when publishing the data will be reported by an **InternalError** exception.

- **create_many** (resp. **update_many**, **delete_many**) allows creating (resp. updating, deleting) several instances in a single call. Resulting DDS orders are stopped at the first error (and the **index** of the erroneous instance value is reported in the raised **InternalError** exception).

  If the attribute **is_coherent_write** is **TRUE**, the resulting successful write or dispose DDS orders are placed between a DDS **begin_coherent_updates** and an **end_coherent_updates**.
- **create_one** and **create_many** operations check that the targeted instances are not existing prior to the call. This check is performed locally to the component if the attribute **is_global_scope** is **FALSE** or globally to the data space if **is_global_scope** is **TRUE**. In any case, this check is performed before any attempt ordering DDS to write and is applied to all the submitted instances. All the erroneous instances are reported in the **AlreadyCreated** exception (by means of their index in the submitted sequence).
- **update_one** and **update_many** operations check that the targeted instances are existing prior to the call. This check is performed locally to the component if the attribute **is_global_scope** is **FALSE**, or globally to the data space if **is_global_scope** is **TRUE**. In any case, this check is performed before any attempt ordering DDS to write and is applied to all the submitted instances. All the erroneous instances are reported in the **NonExistent** exception (by means of their index in the submitted sequence).
- **delete_one** and **delete_many** operations check that the targeted instances are existing prior to the call. This check is performed locally to the component if the attribute **is_global_scope** is **FALSE**, or globally to the data space if **is_global_scope** is **TRUE**. In any case, this check is performed before any attempt ordering DDS to dispose and is applied to all the submitted instances. All the erroneous instances are reported in the **NonExistent** exception (by means of their index in the submitted sequence).

Note – Global checks may require an attempt to get the instance under the scene and cannot be a full guarantee as a write or a dispose from another participant may always occur between the check and the actual write or dispose. Therefore this setting should be restricted to architectures where a single writer is involved.

Note – In case of a single operation (**create_one**, **update_one**, or **delete_one**) failing on the life cycle check, the sequence parameter of the exception (**AlreadyExisting** or **NonExistent**) will contain 0.

7.2.2.1.2 Data Access - Subscribing Side

Preamble: for all the following operations, **read** means implicitly “with no wait” and **get** means implicitly “with wait.”

Several interfaces allow retrieving data values from DDS data readers:

- A **Reader** allows reading one or several instance values on a given topic according to a given criterion, with no wait.

In addition, the following interfaces allow getting fresh values from a given topic:

- A **Getter** allows getting them in pull mode. It may block to get the proper information.
- A **Listener** allows getting them in push mode, regardless of the instance status.
- A **StateListener** allows getting them in push mode when the instance status is a concern: different operations will be triggered according to the instance status.

The following IDL declarations for those interfaces and related types, are followed by explanations when needed:

Related Types

```
enum AccessStatus {
    FRESH_INFO,
    ALREADY_SEEN
    };

enum InstanceStatus {
    INSTANCE_CREATED,
    INSTANCE_FILTERED_IN,
    INSTANCE_UPDATED,
    INSTANCE_FILTERED_OUT,
    INSTANCE_DELETED
    };

struct ReadInfo {DDS::InstanceHandle_t      instance_handle;
    DDS::Time_t                             source_timestamp;
    AccessStatus                            access_status;
    InstanceStatus                          instance_status;
    };

typedef sequence<ReadInfo> ReadInfoSeq;
```

ReadInfo is the simplified version of DDS **SampleInfo**. Each read or gotten piece of data is accompanied with a **ReadInfo** that specifies:

- The DDS **instance_handle**
- The DDS **source_timestamp**
- Whether the value has already been seen or not by the component (**access_status**)
- The instance status (**instance_status**) at the time of the sample. This status can be:
    - **INSTANCE_CREATED** if this is the first time that the component sees that instance (the instance is then existing for the component).
    - **INSTANCE_FILTERED_IN** if an existing instance reenters the filter after having been filtered out.
    - **INSTANCE_ UPDATED** if an existing instance is modified and stays within the filter.
    - **INSTANCE_FILTERED_OUT** if an existing instance just stopped passing the filter.
    - **INSTANCE_DELETED** if the instance just stopped existing.

The **instance_status** is therefore a combination of several fields in the original DDS **SampleInfo**. Unfortunately, in the current DDS, the fact that a data is filtered out is not reported. However as this is likely to change soon, the two statuses **INSTANCE_FILTERED_IN** and **INSTANCE_FILTERED_OUT** have been added for provision. As long as this feature is not available in DDS, a compliant implementation of this specification is not required to deliver those two statuses.

The following figure shows how the three other values can be computed based on DDS returned information.

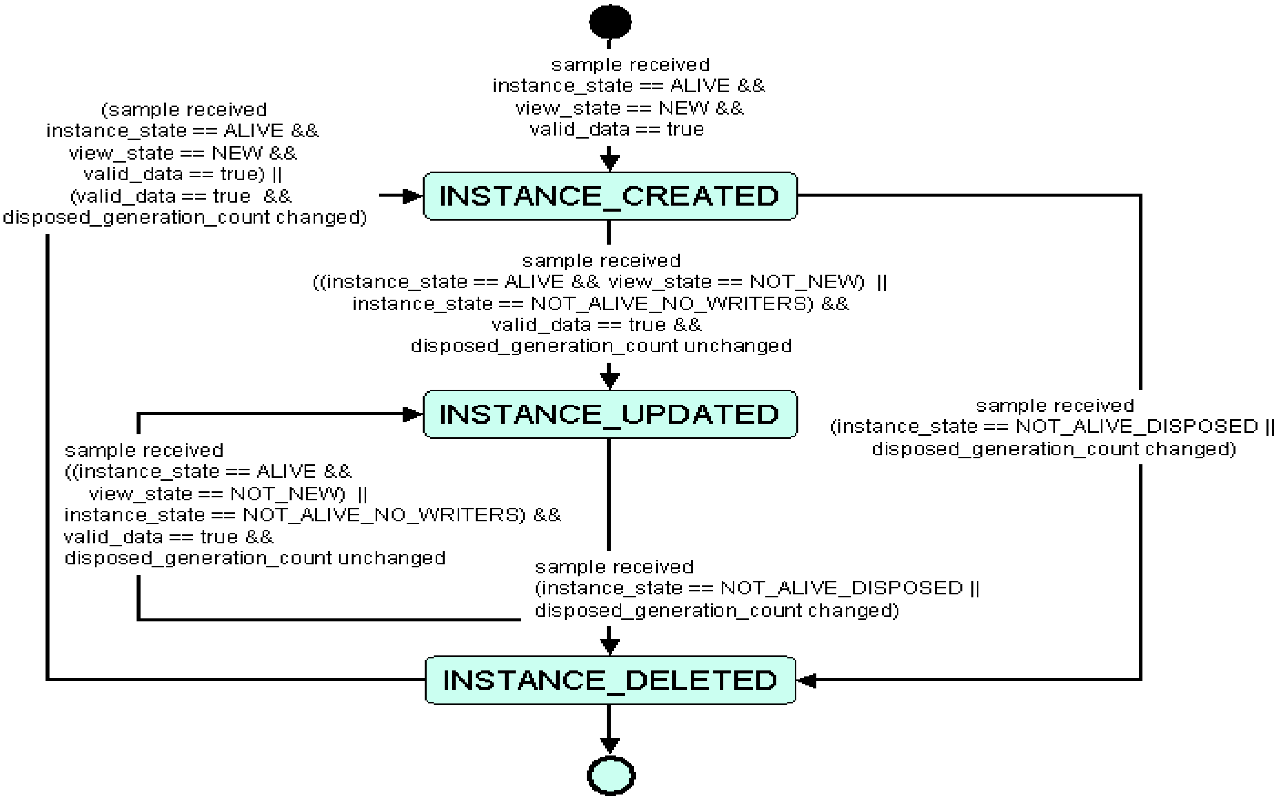

sample received
instance_state == ALIVE &&
view_state == NEW &&
valid_data == true
(sample received
instance_state == ALIVE &&
view_state == NEW &&
valid_data == true) ||
(valid_data == true &&
disposed_generation_count changed)
INSTANCE_CREATED
sample received
((instance_state == ALIVE && view_state == NOT_NEW) ||
instance_state == NOT_ALIVE_NO_WRITERS) &&
valid_data == true &&
disposed_generation_count unchanged
INSTANCE_UPDATED
sample received
(instance_state == NOT_ALIVE_DISPOSED ||
disposed_generation_count changed)
sample received
((instance_state == ALIVE &&
view_state == NOT_NEW) ||
instance_state == NOT_ALIVE_NO_WRITERS) &&
valid_data == true &&
disposed_generation_count unchanged
sample received
(instance_state == NOT_ALIVE_DISPOSED ||
disposed_generation_count changed)
INSTANCE_DELETED

Figure 7.2 - ReadInfo::instance_status State Chart

Note – Except if the **instance_status** is **INSTANCE_DELETED**, the associated data value is valid (other cases where **DDS::SampleInfo::valid_data** would be **FALSE** should be managed by the connector fragment and shouldn't be passed to the component).

Note – When several values are returned, they may be different samples of the same or of different instances. They will always be ordered by instances (i.e., all the samples of the first instance, followed by all the samples of the second one…).

```
struct QueryFilter {
    string              expression;
    DDS::StringSeq      parameters
    };
```

QueryFilter gathers in a single structure a query expression and its related parameters. The **QueryFilter** attribute placed on the **Reader** interface acts as a filter for all the read operations made through a port where such a **Reader** is attached. An empty string **expression** means no query.

This query expression and its related parameters are for DDS use and must comply with DDS rules (see the DDS specification for more details). Any attempt to set the attribute with values that are not accepted by DDS will result in an **InternalError** exception.

Interface Reader

```
local interface Reader  {
        void read_last (inout TSeq data, inout ReadInfoSeq infos)
                raises (InternalError);
        void read_all (inout TSeq data, inout ReadInfoSeq infos)
                raises (InternalError);
        void read_one_last (inout T datum, out ReadInfo info,
                        in DDS::InstanceHandle_t instance_handle)
                raises (NonExistent,
                        InternalError);
        void read_one_all (in T datum, inout TSeq data, inout ReadInfoSeq infos,
                        in DDS::InstanceHandle_t instance_handle)
                raises (NonExistent,
                        InternalError);
        attribute QueryFilter query
                setraises (InternalError);
        };
```

Behavior of a **Reader** is as follows:

- Underlying DDS read operations will be performed with the following DDS access parameters:
    - **SampleStateMask**: **READ** or **NO_READ**
    - **ViewStateMask**: **NEW** or **NOT_NEW**
    - **InstanceStateMask**: **ALIVE**
    - Through the query as specified in the **query** ("" as **expression** means no query).
- **read_last** returns the last sample of all instances. In case of no data, the resulting data will be a void sequence. Any other DDS error when reading the data will be reported by an **InternalError** exception.
- **read_all** returns all samples of all instances. In case of no data, the resulting data will be a void sequence. Any other DDS error when reading the data will be reported by an **InternalError** exception.
- **read_one_last** returns the last sample of a given instance. The targeted instance is designated by the passed instance handle (**instance_handle**) if not **DDS::HANDLE_NIL** or else by the key value in the passed data (**datum**). If a valid handle is passed, it must be in accordance with the key value of the passed data; otherwise, an **InternalError** exception is raised with the returned DDS error code. More generally, any DDS error when reading the data will be reported by an **InternalError** exception.
    - In case the instance does not exist (no data are registered for that instance in DDS), the exception **NonExistent** is raised.
    - In case of a keyless topic, the last value in the topic will be returned as DDS considers all values in such a topic as samples of one unique instance.
- **read_one_all** returns all the samples of a given instance The targeted instance is designated by the passed instance handle (**instance_handle**) if not **DDS::HANDLE_NIL** or else by the key value in the passed data (**datum**). If a valid handle is passed, it must be in accordance with the key value of the passed data; otherwise, an **InternalError** exception is raised with the returned DDS error code. More generally, any DDS error when reading the data will be reported by an **InternalError** exception.

- In case the instance does not exist (no data are registered for that instance in DDS), the exception **NonExistent** is raised.
- In case of a keyless topic, all values will be returned as DDS considers all values in such a topic as samples of one unique instance.

Note – This interface is the basis for a passive data reader (i.e., a component that just looks at the data as they are). It is also very useful for the reactive data getters (i.e., components that need to react to new data, whether they choose to get them in pull mode or be notified in push mode) in their initialization phase. This is the reason why all the DDS ports on the subscribing side will embed a **Reader** basic port.

Interface Getter

```
local interface Getter {
    boolean get_one (out T datum, out ReadInfo info)
        raises (InternalError);
    boolean get_many (inout TSeq data, inout ReadInfoSeq infos)
        raises (InternalError);
    attribute DDS::Duration_ttime_out;
    attribute DataNumber_t max_delivered_data;// default 0 (no limit)
    };
```

Behavior of a **Getter** is as follows:

- **Get** operations are meant to provide information that has not been previously communicated to the participant. They may wait until fresh information is available and are performed with the following parameters:
  - **SampleStateMask**: **NO_READ**
  - **ViewStateMask**: **NEW** or **NOT_NEW**
  - **InstanceStateMask**: **ALIVE** or **NOT_ALIVE**
  - Through the query (if any) of the **Reader** associated to the port
  - Within the time limit specified in **time_out**.
- They all return a **boolean** as result indicating whether actual data are provided (**TRUE**) or if the time-out occurred (**FALSE**).
- **get_one** returns the next sample to be gotten.
- **get_many** returns all the available samples within the limits set by the attribute **max_delivered_data**. In case there are too many available samples, only the first **max_delivered_data** is returned, the others remaining available for a subsequent call. The default value for that attribute is **UNLIMITED** (0).

Interface Listener

```
local interface Listener {
    void on_one_data (in T datum, in ReadInfo info);
    void on_many_data (in TSeq data, in ReadInfoSeq infos);
    };
```

Behavior of a **Listener** is as follows:

- The semantics of **on_one_data** is similar to the one of **Getter::get_one**, except that it is in push mode instead of pull mode.
- The semantics of **on_many_data** is similar to the one of **Getter::get_many**, except that it is in push mode instead of pull mode.
- The operations are called according to the listener **mode** as set in the associated **DataListenerControl** (see Section 7.2.2.1.3, “Data Listener Control,” on page 15). The mode can be:
  - **NOT_ENABLED**: none of these operations are called.
  - **ONE_BY_ONE**: the data are delivered one sample at a time through the on_one_data_operation.
  - **MANY_BY_MANY**: the data are delivered, through the **on_many_data** operation, by groups of samples, according to the **max_delivered_data** limit set in the associated **DataListenerControl**.
- Query filter (if any) will be found in the associated **Reader**.

Interface StateListener

```
local interface StateListener {
    void on_creation (in T datum, in ReadInfo info);
    void on_one_update (in T datum, in ReadInfo info);
    void on_many_updates (in TSeq data, in ReadInfoSeq infos);
    void on_deletion (in T datum, in ReadInfo info);
    };
```

Behavior of a **StateListener** is as follows:

- No operation is called if the **mode** of the associated **StateListenerControl** is **NOT_ENABLED**.
- **on_creation** is triggered if the instance is considered as new in the component scope. Note that in case there is a filter in the **Reader** associated to the port and the attribute **is_filter_interpreted** of the listener control is **TRUE**, this gathers also the case when the instance is filtered in.
- **on_deletion** is triggered if the instance is no more existing. Note that in case there is a filter in the **Reader** associated to the port and the attribute **is_filter_interpreted** of the listener control is **TRUE**, this gathers also the case when the instance is filtered out. The only fields valid in the provided **datum** parameter are the ones that make the key.
- **on_one_update** is triggered if neither **on_creation** nor **on_deletion** apply and the mode of the associated listener control is **ONE_BY_ONE.**
- **on_many_updates** is triggered if neither **on_creation** nor **on_deletion** apply and the mode of the associated listener control is **MANY_BY_MANY.** The number of returned samples is within the limits of the attribute **max_delivered_data** of the associated listener control.
- Query filter (if any) will be found in the associated **Reader**.

7.2.2.1.3 Data Listener Control

The following interface allows controlling the data listener attached to the port to which they are attached. There are two data listener controls:

1. **DataListenerControl**, which embed the basic controlling behavior for any kind of data listeners.
2. **StateListenerControl**, which is a specialization of the former, which add extra features for a **StateListener**.

Interface DataListenerControl

```
enum ListenerMode {
    NOT_ENABLED,
    ONE_BY_ONE,
    MANY_BY_MANY
    };

local interface DataListenerControl {
    attribute ListenerMode    mode;                   // default NOT_ENABLED
    attribute DataNumber_t    max_delivered_data;     // default 0 (no limit)
    };
```

The two attributes of a **DataListenerControl** allows controlling the associated data listener, as follows:

- If the **mode** is **NOT_ENABLED**, the associated listener's operations are not triggered. This is the default setting as it allows the component to perform its initialization phase (likely using the associated **Reader**) before receiving any data notifications.
- If the **mode** is **ONE_BY_ONE**, the unitary operations (i.e., **on_one_data** or **on_one_update**) of the associated listener are triggered.
- If the **mode** is **MANY_BY_MANY**, the grouped operations (i.e., **on_many_data** or **on_many_updates**) of the associated listener are triggered. These operations are called with as many relevant samples as available, possibly limited by the value of **max_delivered_data**. The default value for that attribute is **UNLIMITED** (0).

StateListenerControl

```
local interface StateListenerControl : DataListenerControl {
    attribute boolean     is_filter_interpreted;     // default FALSE
    };
```

This listener control, specific to control a **StateListener**, extends the former **DataListenerControl** with the attribute **is_filter_interpreted**.

- If **TRUE**, the associated listener should consider an instance entering in (resp. going out) the filter (if any) of the related **Reader**, as an instance creation (resp. deletion) and thus trigger the operation **on_creation** (resp. **on_deletion**).
- If **FALSE**, those events should be considered as normal instance updates and thus lead to triggering **on_one_update** or **on_many_updates**, depending on the **mode**.

Note – DDS is not currently reporting that an instance has been filtered out. This behavior has been thus added for provision. A compliant implementation of this specification is not required to support it as long as DDS does not report when instances are filtered out.

7.2.2.1.4 Content Filter Management

In addition to plain topics, DDS provides content-filtered topics for content-based subscriptions. Such a topic has to be created in relation with a classical one and given a filter expression. All data provided by this topic must pass the filter expression. Apart from that characteristic, content-filter topics and classical ones can be used the same way.

The following attribute allows declaring a filter to the port that will be used for DDS content-filtered subscriptions, in case it is given a value at configuration time.

Attribute Filter

```
attribute QueryFilter filter
    setraises (NonChangeable);
```

While the filter expression is immutable and thus can be considered as a structural configuration attribute of a given port, its parameters can be modified dynamically.

The following interface allows changing those parameters.

Interface ContentFilterSetting

```
local interface ContentFilterSetting {
    void set_filter_parameters (in DDS::StringSeq parameters)
        raises (InternalError);
    };
```

7.2.2.1.5 Status Access

DDS is communicating errors or warnings by means of statuses. Some of those statuses are relevant for the component author (e.g., sample lost), others are meaningful system wide (e.g., incompatible QoS) while others carry information that are needed for functioning (e.g., data on readers).

- The first are made available through a **PortStatusListener**; as those statuses may only concern a DDS data reader, a **PortStatusListener** is meaningful only on a DDS port related to subscribing.
- The second are made available through a **ConnectorStatusListener**.
- The last are kept for internal implementation of connectors fragments and therefore not reported.

Interface portStatusListener

```
local interface PortStatusListener {// status that are relevant to the component
    void on_requested_deadline_missed(
        in DDS::DataReader the_reader,
        in DDS::RequestedDeadlineMissedStatus status);
    void on_sample_lost(
        in DDS::DataReader the_reader,
        in DDS::SampleLostStatus status);
    };
```

Interface ConnectorStatusListener

```
local interface ConnectorStatusListener {// status that are relevant system-wide
    void on_inconsistent_topic(
        in DDS::Topic the_topic,
        in DDS::InconsistentTopicStatus status);
    void on_requested_incompatible_qos(
        in DDS::DataReader the_reader,
        in DDS::RequestedIncompatibleQosStatus status);
    void on_sample_rejected(
        in DDS::DataReader the_reader,
        in DDS::SampleRejectedStatus status);
    void on_offered_deadline_missed(
        in DDS::DataWriter the_writer,
        in DDS::OfferedDeadlineMissedStatus status);
    void on_offered_incompatible_qos(
        in DDS::DataWriter the_writer,
        in DDS::OfferedIncompatibleQosStatus status);
    void on_unexpected_status (
        in DDS::Entity the_entity,
        in DDS::StatusKind status_kind);
    };
```

All the operations of those two listeners mimic exactly the related DDS operations, with exactly the same operation name and parameters.

In addition a last operation is added on **ConnectorStatusListener** to report unexpected statuses (**on_unexpected_status**). The two parameters are then the reporting DDS Entity and the DDS status kind.

7.2.2.2 DDS-DCPS Extended Ports

All the interfaces presented in the previous section, can be considered as building blocks to be assembled to form the extended ports. The following are defined:

```
porttype DDS_Write {
    uses Writer                     data;
    uses DDS::DataWriter            dds_entity;
    };

porttype DDS_Update {
    uses Updater                    data;
    uses DDS::DataWriter            dds_entity;
    };

porttype DDS_Read {
    uses Reader                     data;
    attribute QueryFilter           filter
        setraises(NonChangeable);
    uses ContentFilterSetting       filter_config;
    uses DDS::DataReader            dds_entity;
    provides PortStatusListener     status;
    };
```

```
porttype DDS_Get {
    uses Reader                         data;
    uses Getter                         fresh_data;
    attribute QueryFilter               filter
        setraises(NonChangeable);
    uses ContentFilterSetting           filter_config;
    uses DDS::DataReader                dds_entity;
    provides PortStatusListener         status;
    };

porttype DDS_Listen {
    uses Reader                         data;
    uses DataListenerControl            data_control;
    provides Listener                   data_listener;
    attribute QueryFilter               filter
        setraises(NonChangeable);
    uses ContentFilterSetting           filter_config;
    uses DDS::DataReader                dds_entity;
    provides PortStatusListener         status;
    };

porttype DDS_StateListen {
    uses Reader                         data;
    uses StateListenerControl           data_control;
    provides StateListener              data_listener;
    attribute QueryFilter               filter
        setraises(NonChangeable);
    uses ContentFilterSetting           filter_config;
    uses DDS::DataReader                dds_entity;
    provides PortStatusListener         status;
    };
```

All DDS ports combine at least a basic port to access data with a basic port to access underlying DDS entity. **DDS_Get**, **DDS_Listen**, and **DDS_StateListen** split the data access functionality in two ports; the first (**Reader**) is there to set the read criterion and provide operations for the initialization phase, while the second (**Getter**, **Listener**, or **StateListener**) is rather intended to be used in the application processing loop. All the ports intended for the subscribing side comprise also a configuration attribute (**filter**) to set the content filter, a basic port to change the parameters of the filter expression (**filter_config**), and a port to be notified of the relevant statuses (**status**).

7.3 DDS-DCPS Connectors

DDS-DCPS connectors are intended to gather the connector fragments for all possible roles in a given DDS use pattern.

They come with several DDS-DCPS supported ports (which are expressed in the connector as mirror ports), each of them corresponding to a given role within this pattern as well as with related DDS entities and QoS setting.

As DDS-DCPS ports, DDS-DCPS connectors are parameterized by a data type. As they are very similar to components (from the D&C standpoint), they have configuration properties that allow specifying all the elements that are needed to properly instantiate them, namely:

- The name of the DDS Topic that is associated to the data type.
- The list of fields making up the key for that Topic.

- The DDS Domain Id.
- The QoS settings that are to be applied to the underlying DDS entities (how these settings are expressed is explained in Section 7.4, “Configuration and QoS Support,” on page 21).

Having all this information gathered at the connector-level (rather than split in each DDS participants) gives the ability to better master system consistency.

In addition, they provide a port to report configuration errors (e.g., to be used i.e., by a supervision service).

7.3.1 Base Connectors

DDS_Base connector uses a **ConnectorStatusListener** port for reporting configuration errors and contains attributes to store the Domain identifier and the QoS profile (see 7.4.2, ’DDS QoS Policies in XML’ for more details on QoS profile). The QoS profile could be given either as a file URL or as the XML string itself.

Any attempt to change those attributes once the configuration is complete will raise a **NonChangeable** exception.

All DDS connectors should inherit from that base.

```
connector DDS_Base {
    uses ConnectorStatusListener     error_listener;
    attribute DDS:DomainId_t         domain_id
        setraises (NonChangeable);
    attribute string                 qos_profile          // File URL or XML string
        setraises (NonChangeable);
    };
```

DDS_TopicBase extends the **DDS_Base** with the name of one topic and its key description. **DDS_TopicBase** should be the base for all mono-topic connectors.

```
connector DDS_TopicBase : DDS_Base {
    attribute string                 topic_name
        setraises (NonChangeable);
     attribute DDS::StringSeq        key_fields
        setraises (NonChangeable);
    };
```

As the attributes of **DDS_Base**, the attributes of **DDS_TopicBase** are also non changeable once configured. Any attempt to change them once the configuration is complete will raise a **NonChangeable** exception.

7.3.2 Pattern State Transfer

This pattern corresponds to participants that publish the state of data they manage (role **observable**), associated with other participants that subscribe to get the information (role **observer**). All those roles relate to the connector’s topic.

Observers can be of various kinds:

- **passive_observer** are just reading the state when they want.
- **pull_observer** are getting the state changes.
- **push_observer** are being notified with the state changes.

- **push_state_observer** are being notified with the state changes with different operations depending on the instance status.

The connector definition is as follows:

```
connector DDS_State : DDS_TopicBase {
    mirrorport DDS_Update           observable;
    mirrorport DDS_Read             passive_observer;
    mirrorport DDS_Get              pull_observer;
    mirrorport DDS_Listen           push_observer;

    mirrorport DDS_StateListen      push_state_observer;
    };
```

Typically, with this pattern, **HISTORY QoS** should be set to **KEEP_LAST**.

7.3.3 Pattern Event Transfer

This pattern corresponds to participants sending events over DDS (role **supplier**), while others consume them (role **consumer**). All those roles relate to the connector's topic.

Consumers can be of various kinds:

- **pull_consumer** are getting the events.
- **push_consumer** are being notified with the events.

The connector definition is as follows:

```
connector DDS_Event : DDS_TopicBase {
    mirrorport DDS_Write            supplier;
    mirrorport DDS_Get              pull_consumer;
    mirrorport DDS_Listen           push_consumer;
    };
```

Typically, with this pattern, **HISTORY QoS** should be set to **KEEP_ALL**.

7.4 Configuration and QoS Support

7.4.1 DCPS Entities

When the connector fragments are deployed, they must create under the scene the DDS entities that are needed to get the wanted interaction.

As they are defined, the DDS ports are related to one data type and should therefore be attached to one **DataReader** and/or **DataWriter**, which are entirely dedicated to their port.

The allocation rule for the **Subscriber**, **Publisher**, and **DomainParticipant** is less straightforward as they may be allocated to the port or to the component (meaning that they will be shared by the ports of that component) or to the container (meaning that they will be shared by the components running in that container). Consequently, even if the QoS requirements are expressed on a port basis, components and containers can be given DDS entities that can be used by the infrastructure for servicing embedded ports if they meet the port requirements.

7.4.2 DDS QoS Policies in XML

To ease the consistent management of DDS QoS settings, this specification defines *QoS profiles*. A QoS profile takes the form of an XML string and can gather *QoS*[1] for several DDS entities that form a whole.

The following sections explain how to build QoS Profiles in XML. The XML Schema as well as a QoS Profile with all default values QoS policies, as specified in [DDS], are in Annex C and Annex D respectively.

7.4.2.1 XML File Syntax

The XML configuration file must follow these syntax rules:

- The syntax is XML and the character encoding is UTF-8.
- Opening tags are enclosed in **<>**; closing tags are enclosed in **</>**.
- A value is a UTF-8 encoded string. Legal values are alphanumeric characters. All leading and trailing spaces are removed from the string before it is processed.
  For example, "**<tag> value </tag>**" is the same as "**<tag>value</tag>**."
- All values are case-sensitive unless otherwise stated.
- Comments are enclosed as follows: **<!-- comment -->**.
- The root tag of the configuration file must be **<dds>** and end with **</dds>**.

The primitive types for tag values are specified in the following table:

Table 7.1 QoS Profile: Supported Tag Values

Type	Format	Notes
Boolean	**yes**, **1**, **true** or **BOOLEAN_TRUE**:	Not case-sensitive
	no, **0**, **false** or **BOOLEAN_FALSE**:	
Enum	A string. Legal values are the ones defined for QoS Policies in the DCPS IDL of DDS specification [DDS]	Must be specified as a string. (Do not use numeric values.)
Long	**-2147483648** to **2147483647** or **0x80000000** to **0x7fffffff** or **LENGTH_UNLIMITED**	A 32-bit signed integer
UnsignedLong	**0** to **4294967296** or **0** to **0xffffffff**	A 32-bit unsigned integer

7.4.2.2 Entity QoS

To configure the QoS for a DDS Entity using XML, the following tags have to be used:

- **<participant_qos>**
- **<publisher_qos>**
- **<subscriber_qos>**

[1] A QoS is the set of QoS policies for a given DDS entity (DataReader, DataWriter...)

- **<topic_qos>**
- **<datawriter_qos>**
- **<datareader_qos>**

Each QoS is identified by a name. The QoS can inherit its values from other QoSs described in the XML file. For example:

```
<datawriter_qos name="DerivedWriterQos" base_name="BaseWriterQos">
    <history>
        <kind>KEEP_ALL_HISTORY_QOS</kind>
    </history>
</datawriter_qos>
```

In the above example, the writer QoS named ‘**DerivedWriterQos**’ inherits the values from the writer QoS **‘BaseWriterQos.’** The **HistoryQosPolicy** kind is set to **KEEP_ALL_HISTORY_QOS**.

Each XML tag with an associated name can be uniquely identified by its fully qualified name in C++ style. The writer, reader, and topic QoSs can also contain an attribute called **topic_filter** that will be used to associate a set of topics to a specific QoS when that QoS is part of a DDS profile. See Section 7.4.2.3.2, “Topic Filters,” on page 26.

7.4.2.2.1 QoS Policies

The fields in a **QosPolicy** are described in XML using a 1-to-1 mapping with the equivalent IDL representation in the DDS specification [DDS]. For example, the **Reliability QosPolicy** is represented with the following structures:

```
struct Duration_t {
    long sec;
    unsigned long nanosec;
    };

struct ReliabilityQosPolicy {
    ReliabilityQosPolicyKind kind;
    Duration_t max_blocking_time;
    };
```

The equivalent representation in XML is as follows:

```
<reliability>
    <kind></kind>
    <max_blocking_time>
        <sec></sec>
        <nanosec></nanosec>
    </max_blocking_time>
</reliability>
```

7.4.2.2.2 Sequences

In general, the sequences contained in the QoS policies are described with the following XML format:

```
<a_sequence_member_name>
    <element>...</element>
    <element>...</element>
    …
</a_sequence_member_name>
```

Each element of the sequence is enclosed in an **<element>** tag., as shown in the following example:

```
property>
    <value>
        <element>
            <name>my name</name>
            <value>my value</value>
        </element>
        <element>
            <name>my name2</name>
            <value>my value2</value>
        </element>
    </value>
</property>
```

A sequence without elements represents a sequence of length 0. For example:

```
<a_sequence_member_name/>
```

As a special case, sequences of octets are represented with a single XML tag enclosing a sequence of decimal / hexadecimal values between 0..255 separated with commas. For example:

```
<user_data>
    <value>100,200,0,0,0,223</value>
</user_data>
<topic_data>
    <value>0xff,0x00,0x8e,0xEE,0x78</value>
</topic_data>
```

7.4.2.2.3 Arrays

In general, the arrays contained in the QoS policies are described with the following XML format:

```
<an_array_member_name>
    <element>...</element>
    <element>...</element>
    ...
</an_array_member_name>
```

Each element of the array is enclosed in an **<element>** tag.

As a special case, arrays of octets are represented with a single XML tag enclosing an array of decimal/hexadecimal values between 0..255 separated with commas. For example:

```
<datareader_qos>
    ...
    <user_data>
        <value>100,200,0,0,0,223</value>
    </user_data>
</datareader_qos>
```

7.4.2.2.4 Enumeration Values

Enumeration values are represented using their IDL string representation. For example:

```
<history>
    <kind>KEEP_ALL_HISTORY_QOS</kind>
</history>
```

7.4.2.2.5 Time Values (Durations)

Following values can be used for fields that require seconds or nanoseconds:

- DURATION_INFINITE_SEC
- DURATION_ZERO_SEC
- DURATION_INFINITE_NSEC
- DURATION_ZERO_NSEC

The following example shows the use of time values:

```
<deadline>
    <period>
        <sec>DURATION_INFINITE_SEC</sec>
        <nanosec>DURATION_INFINITE_NSEC</nanosec>
    </period>
</deadline>
```

7.4.2.3 QoS Profiles

A QoS profile groups a set of related QoS, usually one per entity. For example:

```
<qos_profile name="StrictReliableCommunicationProfile">
    <datawriter_qos>
        <history>
            <kind>KEEP_ALL_HISTORY_QOS</kind>
        </history>
        <reliability>
            <kind>RELIABLE_RELIABILITY_QOS</kind>
        </reliability>
    </datawriter_qos>
    <datareader_qos>
        <history>
            <kind>KEEP_ALL_HISTORY_QOS</kind>
        </history>
        <reliability>
            <kind>RELIABLE_RELIABILITY_QOS</kind>
        </reliability>
    </datareader_qos>
</qos_profile>
```

7.4.2.3.1 QoS-Profile Inheritance

A QoS Profile can inherit its values from other QoS Profiles described in the XML file using the tag **base_name**. For example:

```
<qos_profile name="MyProfile" base_name="BaseProfile">
    ...
</qos_profile>
```

A QoS profile cannot inherit from other QoS profiles if the last one has not been parsed before.

7.4.2.3.2 Topic Filters

A QoS profile may contain several writer, reader, and topic QoSs that can be selected based on the evaluation of a filter expression on the topic name.

The filter expression is specified as an attribute in the XML QoS definition thanks to a **topic_filter** tag. For example:

```
<qos_profile name="StrictReliableCommunicationProfile">
    <datawriter_qos topic_filter="A*">
        <history>
            <kind>KEEP_ALL_HISTORY_QOS</kind>
        </history>
        <reliability>
            <kind>RELIABLE_RELIABILITY_QOS</kind>
        </reliability>
    </datawriter_qos>
    <datawriter_qos topic_filter="B*">
        <history>
            <kind>KEEP_ALL_HISTORY_QOS</kind>
        </history>
        <reliability>
            <kind>RELIABLE_RELIABILITY_QOS</kind>
        </reliability>
        <resource_limits>
            <max_samples>128</max_samples>
            <max_samples_per_instance>128</max_samples_per_instance>
            <initial_samples>128</initial_samples>
            <max_instances>1</max_instances>
            <initial_instances>1</initial_instances>
        </resource_limits>
    </datawriter_qos>
    …
</qos_profile>
```

If **topic_filter** is not specified, the filter '*' will be assumed. The QoSs with an explicit **topic_filter** attribute definition will be evaluated in order; they have precedence over a QoS without a **topic_filter** expression.

7.4.2.3.3 QoS Profiles with a Single QoS

The definition of an individual QoS is a shortcut for defining a QoS profile with a single QoS. For example:

```
<datawriter_qos name="KeepAllWriter">
    <history>
        <kind>KEEP_ALL_HISTORY_QOS</kind>
    </history>
</datawriter_qos>
```

is equivalent to the following:

```
<qos_profile name="KeepAllWriter">
    <writer_qos>
        <history>
            <kind>KEEP_ALL_HISTORY_QOS</kind>
        </history>
    </writer_qos>
</qos_profile>
```

7.4.3 Use of QoS Profiles

A QoS Profile shall be attached as a configuration attribute to a DDS connector. This profile should contain all values for initializing DDS Entities that are required by the connector.

In case of the connector involves several topics (which is not the case with the normative DDS-DCPS extended ports and connectors), then the **topic_filter** feature of the QoS Profile may be used to properly allocate values to entities.

A QoS Profile could also be attached to a DDS-capable component (i.e., a component that has at least one DDS port) to define component's default **DomainParticipant**, **Subscriber**, and/or **Publisher**. These default entities should be used preferably if their setting is compatible with the QoS requested in the connector's profile. If they are not compatible, specific entities dedicated to the 'non-compatible' port will be created. In this component profile, any **topic_qos**, **datareader_qos**, or **datawriter_qos** is simply ignored.

In addition, a similar QoS Profile could be attached to a DDS-capable container (i.e., a container hosting DDS-capable components to define container's defaults that should be used in priority, if suitable).

7.4.4 Other Configuration – Threading Policy

As opposed to the DDS QoS policies that need to be managed system-wide, the threading policy is local to the component using a DDS port. The threading policy could be set at several levels:

- port (for all its facets)
- component (for all the facets of its ports)
- container (for all the facets of its components' ports)

When a facet is activated, the threadpool attached to the port:

- if there is no port's policy, the component's threadpool is used.
- if there is no component's one, the container's threadpool is used.
- if there is no container's policy, then the default is applied.

8 DDS-DLRL Extended Ports and Connectors

This chapter instantiates the Generic Interaction Support of CCM, in order to define ports and connectors for DDS-DLRL. This chapter assumes an *a-priori* knowledge of this CCM extension and of DDS specification (in particular of the DLRL part).

The rationale for providing support to DLRL flavor of CCM in CCM is very similar to the one that drives the DCPS support, namely simplify the use and enforce separation of concerns.

The DLRL principles have been to ease as much as possible the publication and reception of data by providing ability to define plain application objects whose some data members are mapped to DDS topics. Then plain object manipulation (creation, update, deletion) is automatically translated under the scene by the DLRL layer in DCPS publications, while similarly DCPS receptions are automatically turned in updating objects. This interface is very developer-friendly and can hardly be simplified.

In return, according to CCM principles, the setting of the DLRL infrastructure, namely the creation of the Cache and of the Object Homes, their registration as well as the adjustment if needed of the DCPS entities QoS (all this making up the DLRL configuration) can be put apart from the application code.

The design principles to identify DLRL ports and connectors is identical to DCPS application, in that:

- Ports will capture programming contracts for components.
- Connectors will be the support for system-wide configuration.

8.1 Design Principles

8.1.1 Scope of DLRL Extended Ports

In DLRL, the natural entry point to deal with objects of a given type is the related **ObjectHome** and all objects of a given **Cache** are very related and need to be managed consistently.

Consequently, a DLRL extended port should be created to give access to all objects of a given Cache. That extended port will contain one **receptacle** for each **ObjectHome** and another **receptacle** for the **Cache** functional operations (i.e., excluding all the operations that are related to configuration that will be for the only use of the **Connector** implementation).

8.1.2 Scope of DLRL Connectors

A connector is the natural support to gather all the DLRL extended ports that are related to the same set of topics in order to master their configuration system-wide.

As potentially a DLRL object model (consistent set of DLRL classes and their relations) is specific to one participant, it could be as many DLRL extended ports as participants sharing the same set of DCPS topics. However, nothing prevents deploying several components using the same DLRL object model (therefore using the same extended port definition).

8.2 DDS-DLRL Extended Ports

Due to its essential variable composition, it is not possible to define one normative DLRL extended port. In return, the definition of their basic ports as well as the extended port composition rule are normative.

8.2.1 DLRL Basic Ports

8.2.1.1 Cache Operation

This interface is intended to type the **receptacle** dedicated to using the **Cache** once initialized by the infrastructure. It therefore contains only the operative subset of the **DDS::Cache** functions and attributes.

All the retained functions mimic exactly the **DDS::Cache** ones, and therefore request the same parameters and return the same result. Similarly, all the retained attributes are identical to the **DDS::Cache** ones.

```
local interface CacheOperation {
    // Cache kind
    // ----------
    readonly attribute DDS::CacheUsage           cache_usage;

    // Other Cache attributes
    // ---------------------
    readonly attribute DDS::ObjectRootSeq        objects;
    readonly attribute boolean                   updates_enabled;
    readonly attribute DDS::ObjectHomeSeq        homes;
    readonly attribute DDS::CacheAccessSeq       sub_accesses;
    readonly attribute DDS::CacheListenerSeq     listeners;

    // Cache update
    // ------------
    void DDS::refresh( )
        raises (DDS::DCPSError);

    // Listener management
    // -------------------
    void attach_listener (in DDS::CacheListener listener);
    void detach_listener (in DDS::CacheListener listener);

    // Updates management
    // ------------------
    void enable_updates ();
    void disable_updates ();

    // CacheAccess Management
    // ---------------------
    DDS::CacheAccess create_access (in DDS::CacheUsage purpose)
        raises (DDS::PreconditionNotMet);
    void delete_access (in DDS::CacheAccess access)
        raises (DDS::PreconditionNotMet);
    };
```

8.2.1.2 DLRL Class (ObjectHome)

For each DLRL object type to be part of the application, the DLRL extended port should comprise a **receptacle** of type the related home inheriting from **DDS::ObjectHome**. That class should have been generated by the DDS-DLRL product tooling.

All accesses to the DLRL objects of this type will be manageable through this entry point.

8.2.2 DLRL Extended Ports Composition Rule

DLRL extended ports are as many as applications. A DLRL extended port should be made of:

- A **CacheOperation** receptacle
- As many **DDS:ObjectHome**-derived receptacles as DLRL object types that will be used by the component using that DLRL port (those types having been generated by the DDS-DLRL product tooling).

Following is an example of such a declaration:

```
porttype MyDlrlPort_1 {
    uses CCM_DDS::CacheOperationcache;
    uses FooHome          foo_home;              // entry point for Foo objects
    uses BarHome          bar_home               // entry point for Bar objects
    };
```

Based on this information, the related connector fragment will, under the scene:

- Create the cache according to the specified **CacheOperation::cache_usage**.
- Instantiate and register the specified **ObjectHome** (that will create the DCPS entities according to the DLRL → DCPS mapping).
- Apply the QoS profile to modify underlying DCPS entities (if specified in the connector).
- Enable the infrastructure so that DLRL objects can be created and used the DLRL way.

8.3 DDS-DLRL Connectors

As a DLRL connector aims at gathering as many mirror ports as there are different object models in the system sharing the related topics, its composition is essentially variable and application-dependent, and a unique standard DLRL connector cannot be defined. A DLRL connector should inherit from the connector **DDS_Base,** to be given a **ConnectorStatusListener** port, a domain id and a QoS profile attribute, and add as many mirror ports as there exist DLRL extended ports to share the related set of topics.

Following is an example of such a declaration:

```
connector MyDlrlConnector : CCM_DDS::DDS_Base {
    mirrorport MyDlrlPort_1 p1;
    mirrorport MyDlrlPort_2 p2;
    mirrorport MyDlrlPort_3 p3;
    };
```

8.4 Configuration and QoS Support

8.4.1 DDS Entities

As a DLRL port corresponds to one **Cache**, it must be given its own **Publisher** and/or **Subscriber** (depending on the cache usage). In addition, it will get as many **DataReaders** and/or **DataWriters** as there are topics used by the DLRL objects.

8.4.2 Use of QoS Profiles

Configuring DLRL ports can be achieved exactly with the same philosophy as for DCPS ports, with the same definition for a QoS Profile (see sections Section 7.4.2, “DDS QoS Policies in XML,” on page 22 and Section 7.4.3, “Use of QoS Profiles,” on page 27), except that, as the QoS Profile attached to the DLRL connector should contain values for all the topics involved, the **topic_filter** feature of the QoS Profile is to be used in case there is a need to specify different QoS values for different topics.

Annex A
IDL3+ of DDS-DCPS Ports and Connectors

(normative)

```
#include "dds_rtf2_dcps.idl"

module CCM_DDS {

    // ===============================================================================
    // Non-typed part
    //         (here are placed all the constructs that are not dependent on the data type)
    // ===============================================================================
    // --------------------------
    // Enums, structs and Typedefs
    // --------------------------
    typedef unsigned long                  DataNumber_t;           // count or index of data
    typedef sequence<DataNumber_t>         DataNumberSeq;

    const DataNumber_t UNLIMITED = 0;

    enum AccessStatus {
            FRESH_INFO,
            ALREADY_SEEN
            };

    enum InstanceStatus {                  // at sample time, as perceived by the component
            INSTANCE_CREATED,
            INSTANCE_FILTERED_IN,
            INSTANCE_UPDATED,
            INSTANCE_FILTERED_OUT,
            INSTANCE_DELETED
            };

    struct ReadInfo {
            DDS::InstanceHandle_t   instance_handle;
            DDS::Time_t             source_timestamp;
            AccessStatus            access_status;
            InstanceStatus          instance_status;
            };
    typedef sequence<ReadInfo> ReadInfoSeq;

    struct QueryFilter {
            string                  expression;
            DDS::StringSeq          parameters;
            };

    // Data Listener control
    // ---------------------
    enum ListenerMode {
            NOT_ENABLED,
            ONE_BY_ONE,
            MANY_BY_MANY
            };

    // ----------
    // Exceptions
    // ----------
    exception AlreadyCreated {
            DataNumberSeq indexes;          // of the erroneous
            };

    exception NonExistent{
            DataNumberSeq indexes;          // of the erroneous
            };

    exception InternalError{
            DDS::ReturnCode_t error_code;   // DDS codes that are relevant:
                                            // ERROR (1);
                                            // UNSUPPORTED (2);
                                            // BAD_PARAMETER (3)
                                            // PRECONDITION_NOT_MET (4)
                                            // OUT_OF_RESOURCE (5)
            DataNumber_t index;             // of the erroneous
            };

    exception NonChangeable {};

    // ----------
    // Interfaces
    // ----------

    // Listener Control
    // ----------------
    local interface DataListenerControl {
            attribute ListenerMode          mode;                   // default NOT_ENABLED
            attribute DataNumber_t          max_delivered_data;     // default 0 (no limit)
            };

    local interface StateListenerControl : DataListenerControl {
            attribute boolean                       is_filter_interpreted;  // default FALSE
            };

    // Content Filter Parameters Setting
    // --------------------------------
    local interface ContentFilterSetting {
            void set_filter_parameters (in DDS::StringSeq parameters)
                    raises (InternalError);
            };

    // Status Access
    // -------------
    local interface PortStatusListener {    // status that are relevant to the component
            void on_requested_deadline_missed(
                    in DDS::DataReader                              the_reader,
                    in DDS::RequestedDeadlineMissedStatus           status);
            void on_sample_lost(
                    in DDS::DataReader                                  the_reader,
                    in DDS::SampleLostStatus                            status);
            };

    local interface ConnectorStatusListener { // status that are relevant system-wide
            void on_inconsistent_topic(
                    in DDS::Topic                                       the_topic,
                    in DDS::InconsistentTopicStatus                     status);
            void on_requested_incompatible_qos(
                    in DDS::DataReader                                  the_reader,
                    in DDS::RequestedIncompatibleQosStatus              status);
            void on_sample_rejected(
                    in DDS::DataReader                                  the_reader,
                    in DDS::SampleRejectedStatus                        status);
            void on_offered_deadline_missed(
                    in DDS::DataWriter                                  The_writer,
                    in DDS::OfferedDeadlineMissedStatus                 status);
            void on_offered_incompatible_qos(
                    in DDS::DataWriter                                  the_writer,
                    in DDS::OfferedIncompatibleQosStatus                status);
            void on_unexpected_status (
                    in DDS::Entity                                      the_entity,
                    in DDS::StatusKind                                  status_kind);
            };

    // ---------------
    // Connector bases
    // ---------------
    connector DDS_Base {
            uses ConnectorStatusListener                        error_listener;
            attribute DDS::DomainId_t                           domain_id
                    setraises (NonChangeable);
            attribute string                                    qos_profile     // File URL or XML string
                    setraises (NonChangeable);
            };

    connector DDS_TopicBase : DDS_Base {
            attribute string                                    topic_name
                    setraises (NonChangeable);
            attribute DDS::StringSeq                            key_fields
                    setraises (NonChangeable);
            };

    // ==============================================================================
    // Typed sub-part
    //      (here are placed all the construct that are depending on the data type
    //       either directly or indirectly)
    // ==============================================================================

    module Typed <typename T, sequence<T> TSeq> {
    // Gathers all the constructs that are dependent on the data type (T),
    // either directly -- interfaces making use of T or TSeq,
    // or indirectly -- porttypes using or providing those intefaces.
    // TSeq is passed as a second parameter to avoid creating a new sequence type.
            // ------------------------------------
            // Interfaces to be 'used' or 'provided'
            // ------------------------------------

            // Data access - publishing side
            // -----------------------------

            // -- InstanceHandle Manager
            local interface InstanceHandleManager {
                    DDS::InstanceHandle_t register_instance (in T datum)
                            raises (InternalError);
                    void unregister_instance (in T datum, in DDS::InstanceHandle_t instance_handle)
                            raises (InternalError);
                    };

            // -- Writer: when the instance lifecycle is not a concern
            local interface Writer : InstanceHandleManager {
                    void write_one (in T datum, in DDS::InstanceHandle_t instance_handle)
                            raises (InternalError);
                    void write_many (in TSeq data)
                            raises (InternalError);
                    attribute boolean is_coherent_write;            // FALSE by default
                    // behavior
                    // ---------
                    // - the handle is exactly managed as by DDS (cf. DDS spec for more details)
                    // - attempt to write_many is stopped at the first error
                    // - if is_coherent_write, DDS write orders issued by a write_many
                    //         are placed between begin/end coherent updates (even if an error occurs)
                    };

            // -- Updater: when the instance lifecycle is a concern
            local interface Updater : InstanceHandleManager {
                    void create_one (in T datum, in DDS::InstanceHandle_t instance_handle)
                            raises (AlreadyCreated,
                                    InternalError);
                    void update_one (in T datum, in DDS::InstanceHandle_t instance_handle)
                            raises (NonExistent,
                                    InternalError);
                    void delete_one (in T datum,in DDS::InstanceHandle_t instance_handle)
                            raises (NonExistent,
                                    InternalError);

                    void create_many (in TSeq data)
                            raises (AlreadyCreated,
                                    InternalError);
                    void update_many (in TSeq data)
                            raises (NonExistent,
                                    InternalError);
                    void delete_many (in TSeq data)
                            raises (NonExistent,
                                    InternalError);

                    readonly attribute boolean is_global_scope;             // FALSE by default
                    attribute boolean is_coherent_write;                    // FALSE by default
```

```
    // behavior
    // --------
    // - the handle is exactly managed as by DDS (cf. DDS spec for more details)
    // - exceptions AlreadyCreated or NonExistent are raised at least if a local
    //      conflict exists; in addition if is_global_scope is true, the test on
    //      existence attempts to take into account the instances created outside
    //          - note: this check requires to previously attempt to read (not free)
    //          - note: this check is not 100% guaranteed as a creation or a deletion
    //            may occur in the short time between the check and the DDS order
    // - For *-many operations:
    //          - global check is performed before actual write or dispose
    //            (in case of error, all the erroneous instances are reported
    //            in the exception)
    //          - attempt to DDS write or dispose is stopped at the first error
    //          - if is_coherent_write, DDS orders resulting from a *_many operation
    //            are placed between begin/end coherent updates (even if an error
    //            occurs)
    };

// Data access - subscribing side
// ------------------------------

// -- Reader: to simply access to the available data (no wait)
local interface Reader {
    void read_last (inout TSeq data, inout ReadInfoSeq infos)
        raises (InternalError);
    void read_all (inout TSeq data, inout ReadInfoSeq infos)
        raises (InternalError);
    void read_one_last (inout T datum, out ReadInfo info,
            in DDS::InstanceHandle_t instance_handle)
        raises (NonExistent,
            InternalError);
    void read_one_all (in T datum, inout TSeq data, inout ReadInfoSeq infos,
            in DDS::InstanceHandle_t instance_handle)
        raises (NonExistent,
            InternalError);
    attribute QueryFilter query
        setraises (InternalError);
    // behavior
    // --------
    // - read operations are performed with the following parameters
    //          - READ or NO_READ
    //          - NEW or NOT_NEW
    //          - ALIVE
    //          - through the query as specified ("" expression means no query)
    // - data returned:
    //          - read_last returns for each living instance, its last sample
    //          - read_all returns all the samples of all instances
    //            ordered by instance first and then by sample
    //          - read_one_last returns the last sample of the given instance
    //          - read_one_all returns all the samples for the given instance
    //      - read_one operations use the instance_handle the same way
    //        the Writer or Updater *_one operations do
    };
```

```
// -- Getter: to get new data (and wait for)
local interface Getter {
        boolean get_one (out T datum, out ReadInfo info)
                raises (InternalError);
        boolean get_many (inout TSeq data, inout ReadInfoSeq infos)
                raises (InternalError);
        attribute DDS::Duration_ttime_out;
        attribute DataNumber_t max_delivered_data;        // default 0 (no limit)
        // behavior
        // --------
        // - get operations are performed with the following parameters
        //              - NO_READ
        //              - NEW or NOT_NEW
        //              - ALIVE or NOT_ALIVE
        //              - through the query as specified in the associated Reader
        //              - within the time limit specified in time_out
        // - all operations returns TRUE if data are provided
        //       or FALSE if time-out occurred
        // - data returned:
        //              - get_one returns each read sample one by one
        //      - get_many returns all available samples within the
        //               max_delivered_data limit
        };

// -- Listener: similar to a Getter but in push mode
local interface Listener {
        void on_one_data (in T datum, in ReadInfo info);
        void on_many_data (in TSeq data, in ReadInfoSeq infos);
        // behavior
        // --------
        // - on_one_data() trigered is the mode of the associated listener control
        //       is ONE_BY_ONE (then similar to a get_one(), except that in push mode
        //       instead of pull mode)
        // - on_many_data() triggered if the listener mode is MANY_BY_MANY (then
        //       similar to get_many() but in push mode)
        // - query filter (if any) in the associated Reader
        };

// -- StateListener: listener to be notified based on the instance lifecycle
local interface StateListener {
        void on_creation (in T datum, in ReadInfo info);
        void on_one_update (in T datum, in ReadInfo info);
        void on_many_updates (in TSeq data, in ReadInfoSeq infos);
        void on_deletion (in T datum, in ReadInfo info);
        // behavior
        // --------
        // - no operations are trigerred if the mode of the associated listener
        //       control is NOT_ENABLED
        // - on_creation() is triggered if the instance is considered as new in the
        //       component scope; note that in case there is a filter and the attribute
        //       is_filter_interpreted of the listener control is TRUE, this gathers also
        //       the case when the instance is filtered-in.
        // - on_deletion() is triggered if the instance is no more existing; note
        //       that in case there is a filter  and the attribute
        //      is_filter_interpreted of the listener control is TRUE, this gathers
        //      also the case when the instance is filtered-out
        // - on_one_update() is trigrered if neither on_creation() nor on_deletion()
        //      are triggered and the mode of the associated listener control is
        //      ONE_BY_ONE
        // - on_many_updates()is triggered if neither on_creation() nor on_deletion()
        //      are triggered and the mode of the associated listener control is
        //      MANY_BY_MANY; the number of returned samples is within the limits of
        //      max_delivered_data attribute of the associated listener control.
        // - query filter (if any) in the associated Reader
        };


// ---------
// DDS Ports
// ---------

porttype DDS_Write {
        uses Writer                     data;
        uses DDS::DataWriter            dds_entity;
        };

porttype DDS_Update {
        uses Updater                    data;
        uses DDS::DataWriter            dds_entity;
        };

porttype DDS_Read {
        uses Reader                     data;
        attribute QueryFilter           filter
                setraises(NonChangeable);
        uses ContentFilterSetting       filter_config;
        uses DDS::DataReader            dds_entity;
        provides PortStatusListener     status;
        };

porttype DDS_Get {
        uses Reader                     data;
        uses Getter                     fresh_data;
        attribute QueryFilter           filter
                setraises(NonChangeable);
        uses ContentFilterSetting       filter_config;
        uses DDS::DataReader            dds_entity;
        provides PortStatusListener     status;
        };

porttype DDS_Listen {
        uses Reader                     data;
        uses DataListenerControl        data_control;
        provides Listener               data_listener;
        attribute QueryFilter           filter
                setraises(NonChangeable);
        uses ContentFilterSetting       filter_config;
        uses DDS::DataReader            dds_entity;
        provides PortStatusListener     status;
```

```
                };

        porttype DDS_StateListen {
                uses Reader                         data;
                uses StateListenerControl           data_control;
                provides StateListener              data_listener;
                attribute QueryFilter               filter
                        setraises(NonChangeable);
                uses ContentFilterSetting           filter_config;
                uses DDS::DataReader                dds_entity;
                provides PortStatusListener         status;
                };

        // ---------------------------
        // Connectors
        // (Correspond to DDS patterns)
        // ---------------------------

        connector DDS_State : DDS_TopicBase {
                mirrorport DDS_Update               observable;
                mirrorport DDS_Read                 passive_observer;
                mirrorport DDS_Get                  pull_observer;
                mirrorport DDS_Listen               push_observer;
                mirrorport DDS_StateListen          push_state_observer;
                };

        connector DDS_Event : DDS_TopicBase {
                mirrorport DDS_Write                supplier;
                mirrorport DDS_Get                  pull_consumer;
                mirrorport DDS_Listen               push_consumer;
                };
        };

};
```

Annex B
IDL for DDS-DLRL Ports and Connectors

(normative)

```
#include "dds_rtf2_dlrl.idl"

module CCM_DDS {

local interface CacheOperation {
        // Cache kind
        // ----------
        readonly attribute DDS::CacheUsage cache_usage;

        // Other Cache attributes
        // ----------------------
        readonly attribute DDS::ObjectRootSeq                    objects;
        readonly attribute boolean           updates_enabled;
        readonly attribute DDS::ObjectHomeSeq                    homes;
        readonly attribute DDS::CacheAccessSeq                   sub_accesses;
        readonly attribute DDS::CacheListenerSeq                 listeners;

        // Cache update
        // ------------
        void refresh( )
                raises (DDS::DCPSError);

        // Listener management
        // -------------------
        void attach_listener (in DDS::CacheListener listener);
        void detach_listener (in DDS::CacheListener listener);

        // Updates management
        // ------------------
        void enable_updates ();
        void disable_updates ();

        // CacheAccess Management
        // ----------------------
        DDS::CacheAccess create_access (in DDS::CacheUsage purpose)
                raises (DDS::PreconditionNotMet);
        void delete_access (in DDS::CacheAccess access)
                raises (DDS::PreconditionNotMet);
        };

};
```

Annex C
XML Schema for QoS Profiles

(normative)

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://www.omg.org/dds/" xmlns:dds="http:/
/www.omg.org/dds/" targetNamespace="http://www.omg.org/dds/" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <!-- definition of simple types -->
  <xs:simpleType name="elementName">
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-zA-Z0-9 ])+"></xs:pattern>
      <!-- <xs:pattern value="^((::)?([a-zA-Z0-9])+(::([a-zA-Z0-9])+)*)$"/> -->
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="topicNameFilter">
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-zA-Z0-9])+"></xs:pattern>
      <!-- <xs:pattern value="^((::)?([a-zA-Z0-9])+(::([a-zA-Z0-9])+)*)$"/> -->
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="destinationOrderKind">
    <xs:restriction base="xs:string">
      <xs:enumeration value="BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS"></xs:enumeration>
      <xs:enumeration value="BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS"></xs:enumeration>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="durabilityKind">
    <xs:restriction base="xs:string">
      <xs:enumeration value="VOLATILE_DURABILITY_QOS"></xs:enumeration>
      <xs:enumeration value="TRANSIENT_LOCAL_DURABILITY_QOS"></xs:enumeration>
      <xs:enumeration value="TRANSIENT_DURABILITY_QOS"></xs:enumeration>
      <xs:enumeration value="PERSISTENT_DURABILITY_QOS"></xs:enumeration>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="historyKind">
    <xs:restriction base="xs:string">
      <xs:enumeration value="KEEP_LAST_HISTORY_QOS"></xs:enumeration>
      <xs:enumeration value="KEEP_ALL_HISTORY_QOS"></xs:enumeration>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="livelinessKind">
    <xs:restriction base="xs:string">
      <xs:enumeration value="AUTOMATIC_LIVELINESS_QOS"></xs:enumeration>
      <xs:enumeration value="MANUAL_BY_PARTICIPANT_LIVELINESS_QOS"></xs:enumeration>
      <xs:enumeration value="MANUAL_BY_TOPIC_LIVELINESS_QOS"></xs:enumeration>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="presentationAccessScopeKind">
    <xs:restriction base="xs:string">
      <xs:enumeration value="INSTANCE_PRESENTATION_QOS"></xs:enumeration>
      <xs:enumeration value="TOPIC_PRESENTATION_QOS"></xs:enumeration>
      <xs:enumeration value="GROUP_PRESENTATION_QOS"></xs:enumeration>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="reliabilityKind">
    <xs:restriction base="xs:string">
      <xs:enumeration value="BEST_EFFORT_RELIABILITY_QOS"></xs:enumeration>
      <xs:enumeration value="RELIABLE_RELIABILITY_QOS"></xs:enumeration>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="ownershipKind">
    <xs:restriction base="xs:string">
      <xs:enumeration value="SHARED_OWNERSHIP_QOS"></xs:enumeration>
      <xs:enumeration value="EXCLUSIVE_OWNERSHIP_QOS"></xs:enumeration>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="nonNegativeInteger_UNLIMITED">
    <xs:restriction base="xs:string">
      <xs:pattern value="(LENGTH_UNLIMITED|([0-9])*)?"></xs:pattern>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="nonNegativeInteger_Duration_SEC">
    <xs:restriction base="xs:string">
      <xs:pattern value="(DURATION_INFINITY|DURATION_INFINITE_SEC|([0-9])*)?"></xs:pattern>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="nonNegativeInteger_Duration_NSEC">
    <xs:restriction base="xs:string">
      <xs:pattern value="(DURATION_INFINITY|DURATION_INFINITE_NSEC|([0-9])*)?"></xs:pattern>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="positiveInteger_UNLIMITED">
    <xs:restriction base="xs:string">
      <xs:pattern value="(LENGTH_UNLIMITED|[1-9]([0-9])*)?"></xs:pattern>
    </xs:restriction>
  </xs:simpleType>
  <!-- definition of named types -->
  <xs:complexType name="duration">
    <xs:all>
      <xs:element name="sec" type="dds:nonNegativeInteger_Duration_SEC" minOccurs="0"></xs:element>
      <xs:element name="nanosec" type="dds:nonNegativeInteger_Duration_NSEC" minOccurs="0"></
xs:element>
    </xs:all>
  </xs:complexType>
  <xs:complexType name="stringSeq">
    <xs:sequence>
      <xs:element name="element" type="xs:string" minOccurs="0" maxOccurs="unbounded"></xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="deadlineQosPolicy">
    <xs:all>
      <xs:element name="period" type="dds:duration" minOccurs="0"></xs:element>
    </xs:all>
  </xs:complexType>
  <xs:complexType name="destinationOrderQosPolicy">
```

```
    <xs:all>
      <xs:element name="kind" type="dds:destinationOrderKind" minOccurs="0"></xs:element>
    </xs:all>
  </xs:complexType>
  <xs:complexType name="durabilityQosPolicy">
    <xs:all>
      <xs:element name="kind" type="dds:durabilityKind" default="VOLATILE_DURABILITY_QOS"
minOccurs="0"></xs:element>
    </xs:all>
  </xs:complexType>
  <xs:complexType name="durabilityServiceQosPolicy">
    <xs:all>
      <xs:element name="service_cleanup_delay" type="dds:duration" minOccurs="0"></xs:element>
      <xs:element name="history_kind" type="dds:historyKind" default="KEEP_LAST_HISTORY_QOS"
minOccurs="0"></xs:element>
      <xs:element name="history_depth" type="xs:positiveInteger" minOccurs="0"></xs:element>
      <xs:element name="max_samples" type="dds:positiveInteger_UNLIMITED" minOccurs="0"></xs:element>
      <xs:element name="max_instances" type="dds:positiveInteger_UNLIMITED" minOccurs="0"></xs:element>
      <xs:element name="max_samples_per_instance" type="dds:positiveInteger_UNLIMITED"
minOccurs="0"></xs:element>
    </xs:all>
  </xs:complexType>
  <xs:complexType name="entityFactoryQosPolicy">
    <xs:all>
      <xs:element name="autoenable_created_entities" type="xs:boolean" default="true" minOccurs="0"></
xs:element>
    </xs:all>
  </xs:complexType>
  <xs:complexType name="groupDataQosPolicy">
    <xs:all>
      <xs:element name="value" type="xs:base64Binary" minOccurs="0"></xs:element>
    </xs:all>
  </xs:complexType>
  <xs:complexType name="historyQosPolicy">
    <xs:all>
      <xs:element name="kind" type="dds:historyKind" default="KEEP_LAST_HISTORY_QOS" minOccurs="0"></
xs:element>
      <xs:element name="depth" type="xs:positiveInteger" default="1" minOccurs="0"></xs:element>
    </xs:all>
  </xs:complexType>
  <xs:complexType name="latencyBudgetQosPolicy">
    <xs:all>
      <xs:element name="duration" type="dds:duration" minOccurs="0"></xs:element>
    </xs:all>
  </xs:complexType>
  <xs:complexType name="lifespanQosPolicy">
    <xs:all>
      <xs:element name="duration" type="dds:duration" minOccurs="0"></xs:element>
    </xs:all>
  </xs:complexType>
  <xs:complexType name="livelinessQosPolicy">
    <xs:all>
      <xs:element name="kind" type="dds:livelinessKind" default="AUTOMATIC_LIVELINESS_QOS"
minOccurs="0"></xs:element>
      <xs:element name="lease_duration" type="dds:duration" minOccurs="0"></xs:element>
```

```
        </xs:all>
    </xs:complexType>
    <xs:complexType name="ownershipQosPolicy">
        <xs:all>
            <xs:element name="kind" type="dds:ownershipKind" minOccurs="0"></xs:element>
        </xs:all>
    </xs:complexType>
    <xs:complexType name="ownershipStrengthQosPolicy">
        <xs:all>
            <xs:element name="value" type="xs:nonNegativeInteger" minOccurs="0"></xs:element>
        </xs:all>
    </xs:complexType>
    <xs:complexType name="partitionQosPolicy">
        <xs:all>
            <xs:element name="name" type="dds:stringSeq" minOccurs="0"></xs:element>
        </xs:all>
    </xs:complexType>
    <xs:complexType name="presentationQosPolicy">
        <xs:all>
            <xs:element name="access_scope" type="dds:presentationAccessScopeKind"
default="INSTANCE_PRESENTATION_QOS" minOccurs="0"></xs:element>
            <xs:element name="coherent_access" type="xs:boolean" default="false" minOccurs="0"></xs:element>
            <xs:element name="ordered_access" type="xs:boolean" default="false" minOccurs="0"></xs:element>
        </xs:all>
    </xs:complexType>
    <xs:complexType name="readerDataLifecycleQosPolicy">
        <xs:all>
            <xs:element name="autopurge_nowriter_samples_delay" type="dds:duration" minOccurs="0"></
xs:element>
            <xs:element name="autopurge_disposed_samples_delay" type="dds:duration" minOccurs="0"></
xs:element>
        </xs:all>
    </xs:complexType>
    <xs:complexType name="reliabilityQosPolicy">
        <xs:all>
            <xs:element name="kind" type="dds:reliabilityKind" minOccurs="0"></xs:element>
            <xs:element name="max_blocking_time" type="dds:duration" minOccurs="0"></xs:element>
        </xs:all>
    </xs:complexType>
    <xs:complexType name="resourceLimitsQosPolicy">
        <xs:all>
            <xs:element name="max_samples" type="dds:positiveInteger_UNLIMITED" minOccurs="0"></xs:element>
            <xs:element name="max_instances" type="dds:positiveInteger_UNLIMITED" minOccurs="0"></xs:element>
            <xs:element name="max_samples_per_instance" type="dds:positiveInteger_UNLIMITED"
minOccurs="0"></xs:element>
            <xs:element name="initial_samples" type="xs:positiveInteger" minOccurs="0"></xs:element>
            <xs:element name="initial_instances" type="xs:positiveInteger" minOccurs="0"></xs:element>
        </xs:all>
    </xs:complexType>
    <xs:complexType name="timeBasedFilterQosPolicy">
        <xs:all>
            <xs:element name="minimum_separation" type="dds:duration" minOccurs="0"></xs:element>
        </xs:all>
    </xs:complexType>
    <xs:complexType name="topicDataQosPolicy">
    <xs:all>
      <xs:element name="value" type="xs:base64Binary" minOccurs="0"></xs:element>
    </xs:all>
  </xs:complexType>
  <xs:complexType name="transportPriorityQosPolicy">
    <xs:all>
      <xs:element name="value" type="xs:nonNegativeInteger" minOccurs="0"></xs:element>
    </xs:all>
  </xs:complexType>
  <!-- userDataQosPolicy uses base64Binary encoding:
    * Allowed characters are all letters: a-z, A-Z,  digits: 0-9, the characters: '+' '/' '=' and ' '
         +,/.=,the plus sign (+), the slash (/), the equals sign (=), and XML whitespace characters.
    * The number of nonwhitespace characters must be divisible by four.
    * Equals signs, which are used as padding, can only appear at the end of the value,
     and there can be zero, one, or two of them.
    * If there are two equals signs, they must be preceded by one of the following characters:
     A, Q, g, w.
    * If there is only one equals sign, it must be preceded by one of the following characters: A, E, I, M, Q, U, Y, c, g,
k, o, s, w, 0, 4, 8.
  -->
  <xs:complexType name="userDataQosPolicy">
    <xs:all>
      <xs:element name="value" type="xs:base64Binary" minOccurs="0"></xs:element>
    </xs:all>
  </xs:complexType>
  <xs:complexType name="writerDataLifecycleQosPolicy">
    <xs:all>
      <xs:element name="autodispose_unregistered_instances" type="xs:boolean" default="true"
minOccurs="0"></xs:element>
    </xs:all>
  </xs:complexType>

  <xs:complexType name="domainparticipantQos">
    <xs:all>
      <xs:element name="user_data" type="dds:userDataQosPolicy" minOccurs="0"></xs:element>
      <xs:element name="entity_factory" type="dds:entityFactoryQosPolicy" minOccurs="0"></xs:element>
    </xs:all>
    <xs:attribute name="name" type="dds:elementName"></xs:attribute>
    <xs:attribute name="base_name" type="dds:elementName"></xs:attribute>
    <xs:attribute name="topic_filter" type="dds:topicNameFilter"></xs:attribute>
  </xs:complexType>
  <xs:complexType name="publisherQos">
    <xs:all>
      <xs:element name="presentation" type="dds:presentationQosPolicy" minOccurs="0"></xs:element>
      <xs:element name="partition" type="dds:partitionQosPolicy" minOccurs="0"></xs:element>
      <xs:element name="group_data" type="dds:groupDataQosPolicy" minOccurs="0"></xs:element>
      <xs:element name="entity_factory" type="dds:entityFactoryQosPolicy" minOccurs="0"></xs:element>
    </xs:all>
    <xs:attribute name="name" type="dds:elementName"></xs:attribute>
    <xs:attribute name="base_name" type="dds:elementName"></xs:attribute>
    <xs:attribute name="topic_filter" type="dds:topicNameFilter"></xs:attribute>
  </xs:complexType>
  <xs:complexType name="subscriberQos">
    <xs:all>
      <xs:element name="presentation" type="dds:presentationQosPolicy" minOccurs="0"></xs:element>
```

```
         <xs:element name="partition" type="dds:partitionQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="group_data" type="dds:groupDataQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="entity_factory" type="dds:entityFactoryQosPolicy" minOccurs="0"></xs:element>
      </xs:all>
      <xs:attribute name="name" type="dds:elementName"></xs:attribute>
      <xs:attribute name="base_name" type="dds:elementName"></xs:attribute>
      <xs:attribute name="topic_filter" type="dds:topicNameFilter"></xs:attribute>
   </xs:complexType>
   <xs:complexType name="topicQos">
      <xs:all>
         <xs:element name="topic_data" type="dds:topicDataQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="durability" type="dds:durabilityQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="durability_service" type="dds:durabilityServiceQosPolicy" minOccurs="0"></
xs:element>
         <xs:element name="deadline" type="dds:deadlineQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="latency_budget" type="dds:latencyBudgetQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="liveliness" type="dds:livelinessQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="reliability" type="dds:reliabilityQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="destination_order" type="dds:destinationOrderQosPolicy" minOccurs="0"></
xs:element>
         <xs:element name="history" type="dds:historyQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="resource_limits" type="dds:resourceLimitsQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="transport_priority" type="dds:transportPriorityQosPolicy" minOccurs="0"></
xs:element>
         <xs:element name="lifespan" type="dds:lifespanQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="ownership" type="dds:ownershipQosPolicy" minOccurs="0"></xs:element>
      </xs:all>
      <xs:attribute name="name" type="dds:elementName"></xs:attribute>
      <xs:attribute name="base_name" type="dds:elementName"></xs:attribute>
      <xs:attribute name="topic_filter" type="dds:topicNameFilter"></xs:attribute>
   </xs:complexType>
   <xs:complexType name="datareaderQos">
      <xs:all>
         <xs:element name="durability" type="dds:durabilityQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="deadline" type="dds:deadlineQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="latency_budget" type="dds:latencyBudgetQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="liveliness" type="dds:livelinessQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="reliability" type="dds:reliabilityQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="destination_order" type="dds:destinationOrderQosPolicy" minOccurs="0"></
xs:element>
         <xs:element name="history" type="dds:historyQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="resource_limits" type="dds:resourceLimitsQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="user_data" type="dds:userDataQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="ownership" type="dds:ownershipQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="time_based_filter" type="dds:timeBasedFilterQosPolicy" minOccurs="0"></
xs:element>
         <xs:element name="reader_data_lifecycle" type="dds:readerDataLifecycleQosPolicy" minOccurs="0"></
xs:element>
      </xs:all>
      <xs:attribute name="name" type="dds:elementName"></xs:attribute>
      <xs:attribute name="base_name" type="dds:elementName"></xs:attribute>
      <xs:attribute name="topic_filter" type="dds:topicNameFilter"></xs:attribute>
   </xs:complexType>
   <xs:complexType name="datawriterQos">
      <xs:all>
```

```
         <xs:element name="durability" type="dds:durabilityQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="durability_service" type="dds:durabilityServiceQosPolicy" minOccurs="0"></
xs:element>
         <xs:element name="deadline" type="dds:deadlineQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="latency_budget" type="dds:latencyBudgetQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="liveliness" type="dds:livelinessQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="reliability" type="dds:reliabilityQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="destination_order" type="dds:destinationOrderQosPolicy" minOccurs="0"></
xs:element>
         <xs:element name="history" type="dds:historyQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="resource_limits" type="dds:resourceLimitsQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="transport_priority" type="dds:transportPriorityQosPolicy" minOccurs="0"></
xs:element>
         <xs:element name="lifespan" type="dds:lifespanQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="user_data" type="dds:userDataQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="ownership" type="dds:ownershipQosPolicy" minOccurs="0"></xs:element>
         <xs:element name="ownership_strength" type="dds:ownershipStrengthQosPolicy" minOccurs="0"></
xs:element>
         <xs:element name="writer_data_lifecycle" type="dds:writerDataLifecycleQosPolicy" minOccurs="0"></
xs:element>
      </xs:all>
      <xs:attribute name="name" type="dds:elementName"></xs:attribute>
      <xs:attribute name="base_name" type="dds:elementName"></xs:attribute>
      <xs:attribute name="topic_filter" type="dds:topicNameFilter"></xs:attribute>
   </xs:complexType>

   <xs:complexType name="domainparticipantQosProfile">
      <xs:complexContent>
         <xs:restriction base="dds:domainparticipantQos">
            <xs:attribute name="name" type="dds:elementName" use="required"></xs:attribute>
         </xs:restriction>
      </xs:complexContent>
   </xs:complexType>
   <xs:complexType name="topicQosProfile">
      <xs:complexContent>
         <xs:restriction base="dds:topicQos">
            <xs:attribute name="name" type="dds:elementName" use="required"></xs:attribute>
         </xs:restriction>
      </xs:complexContent>
   </xs:complexType>
   <xs:complexType name="publisherQosProfile">
      <xs:complexContent>
         <xs:restriction base="dds:publisherQos">
            <xs:attribute name="name" type="dds:elementName" use="required"></xs:attribute>
         </xs:restriction>
      </xs:complexContent>
   </xs:complexType>
   <xs:complexType name="subscriberQosProfile">
      <xs:complexContent>
         <xs:restriction base="dds:subscriberQos">
            <xs:attribute name="name" type="dds:elementName" use="required"></xs:attribute>
         </xs:restriction>
      </xs:complexContent>
   </xs:complexType>
   <xs:complexType name="datawriterQosProfile">
```

```
       <xs:complexContent>
         <xs:restriction base="dds:datawriterQos">
           <xs:attribute name="name" type="dds:elementName" use="required"></xs:attribute>
         </xs:restriction>
       </xs:complexContent>
     </xs:complexType>
     <xs:complexType name="datareaderQosProfile">
       <xs:complexContent>
         <xs:restriction base="dds:datareaderQos">
           <xs:attribute name="name" type="dds:elementName" use="required"></xs:attribute>
         </xs:restriction>
       </xs:complexContent>
     </xs:complexType>

     <xs:complexType name="qosProfile">
       <xs:sequence>
         <xs:choice maxOccurs="unbounded">
           <xs:element name="datareader_qos" type="dds:datareaderQos" minOccurs="0"
   maxOccurs="unbounded"></xs:element>
           <xs:element name="datawriter_qos" type="dds:datawriterQos" minOccurs="0"
   maxOccurs="unbounded"></xs:element>
           <xs:element name="topic_qos" type="dds:topicQos" minOccurs="0" maxOccurs="unbounded"></
   xs:element>
           <xs:element name="domainparticipant_qos" type="dds:domainparticipantQos" minOccurs="0"
   maxOccurs="unbounded"></xs:element>
           <xs:element name="publisher_qos" type="dds:publisherQos" minOccurs="0"
   maxOccurs="unbounded"></xs:element>
           <xs:element name="subscriber_qos" type="dds:subscriberQos" minOccurs="0"
   maxOccurs="unbounded"></xs:element>
         </xs:choice>
       </xs:sequence>
       <xs:attribute name="name" type="dds:elementName" use="required"></xs:attribute>
       <xs:attribute name="base_name" type="dds:elementName"></xs:attribute>
     </xs:complexType>
</xs:schema>
```

Annex D
Default QoS Profile

(non normative)

The following file content is a XML QoS Profile with all default values as specified in DDS.

```
<!--
Data Distribution Service QoS Profile – Default Values
-->
<dds xmlns="http://www.omg.org/dds/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="file://DDS_QoSProfile.xsd">
<qos_profile name=" DDS DefaultQosProfile">
    <datareader_qos>
        <durability>
              <kind>VOLATILE_DURABILITY_QOS</kind>
        </durability>
        <deadline>
            <period>
                <sec>DURATION_INFINITE_SEC</sec>
                <nanosec>DURATION_INFINITE_NSEC</nanosec>
            </period>
        </deadline>
        <latency_budget>
            <duration>
                <sec>0</sec>
                <nanosec>0</nanosec>
            </duration>
        </latency_budget>
        <liveliness>
            <kind>AUTOMATIC_LIVELINESS_QOS</kind>
            <lease_duration>
                <sec>DURATION_INFINITE_SEC</sec>
                <nanosec>DURATION_INFINITE_NSEC</nanosec>
            </lease_duration>
        </liveliness>
        <reliability>
            <kind>BEST_EFFORT_RELIABILITY_QOS</kind>
            <max_blocking_time>
                <sec>0</sec>
                <nanosec>100000000</nanosec>
            </max_blocking_time>
        </reliability>
        <destination_order>
             <kind>BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS</kind>
        </destination_order>
        <history>
            <kind>KEEP_LAST_HISTORY_QOS</kind>
            <depth>1</depth>
        </history>
        <resource_limits>
```

```
            <max_samples>LENGTH_UNLIMITED</max_samples>
            <max_instances>LENGTH_UNLIMITED</max_instances>
            <max_samples_per_instance>LENGTH_UNLIMITED</max_samples_per_instance>
        </resource_limits>
        <user_data>
            <value></value>
        </user_data>
        <ownership>
            <kind>SHARED_OWNERSHIP_QOS</kind>
        </ownership>
        <time_based_filter>
            <minimum_separation>
                <sec>0</sec>
                <nanosec>0</nanosec>
            </minimum_separation>
        </time_based_filter>
        <reader_data_lifecycle>
            <autopurge_nowriter_samples_delay>
                <sec>DURATION_INFINITE_SEC</sec>
                <nanosec>DURATION_INFINITE_NSEC</nanosec>
            </autopurge_nowriter_samples_delay>
            <autopurge_disposed_samples_delay>
                <sec>DURATION_INFINITE_SEC</sec>
                <nanosec>DURATION_INFINITE_NSEC</nanosec>
            </autopurge_disposed_samples_delay>
        </reader_data_lifecycle>
    </datareader_qos>
    <datawriter_qos>
        <durability>
            <kind>VOLATILE_DURABILITY_QOS</kind>
        </durability>
        <durability_service>
            <service_cleanup_delay>
                <sec>0</sec>
                <nanosec>0</nanosec>
            </service_cleanup_delay>
            <history_kind>KEEP_LAST_HISTORY_QOS</history_kind>
            <history_depth>1</history_depth>
            <max_samples>LENGTH_UNLIMITED</max_samples>
            <max_instances>LENGTH_UNLIMITED</max_instances>
            <max_samples_per_instance>LENGTH_UNLIMITED</max_samples_per_instance>
        </durability_service>
        <deadline>
            <period>
                <sec>DURATION_INFINITE_SEC</sec>
                <nanosec>DURATION_INFINITE_NSEC</nanosec>
            </period>
        </deadline>
        <latency_budget>
            <duration>
                <sec>0</sec>
                <nanosec>0</nanosec>
            </duration>
        </latency_budget>
        <liveliness>
            <kind>AUTOMATIC_LIVELINESS_QOS</kind>
            <lease_duration>
                <sec>DURATION_INFINITE_SEC</sec>
                <nanosec>DURATION_INFINITE_NSEC</nanosec>
            </lease_duration>
        </liveliness>
        <reliability>
            <kind>RELIABLE_RELIABILITY_QOS</kind>
            <max_blocking_time>
                <sec>0</sec>
                <nanosec>100000000</nanosec>
            </max_blocking_time>
       </reliability>
       <destination_order>
           <kind>BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS</kind>
        </destination_order>
        <history>
            <kind>KEEP_LAST_HISTORY_QOS</kind>
            <depth>1</depth>
        </history>
        <resource_limits>
            <max_samples>LENGTH_UNLIMITED</max_samples>
            <max_instances>LENGTH_UNLIMITED</max_instances>
            <max_samples_per_instance>LENGTH_UNLIMITED</max_samples_per_instance>
        </resource_limits>
        <transport_priority>
            <value>0</value>
        </transport_priority>
        <lifespan>
            <duration>
                <sec>DURATION_INFINITE_SEC</sec>
                <nanosec>DURATION_INFINITE_NSEC</nanosec>
            </duration>
        </lifespan>
        <user_data>
            <value></value>
       </user_data>
       <ownership>
           <kind>SHARED_OWNERSHIP_QOS</kind>
        </ownership>
        <ownership_strength>
            <value>0</value>
        </ownership_strength>
        <writer_data_lifecycle>
            <autodispose_unregistered_instances>true</autodispose_unregistered_instances>
        </writer_data_lifecycle>
    </datawriter_qos>
    <domainparticipant_qos>
        <user_data>
            <value></value>
        </user_data>
        <entity_factory>
            <autoenable_created_entities>true</autoenable_created_entities>
        </entity_factory>
    </domainparticipant_qos>
```

```
<subscriber_qos>
    <presentation>
        <access_scope>INSTANCE_PRESENTATION_QOS</access_scope>
        <coherent_access>false</coherent_access>
        <ordered_access>false</ordered_access>
    </presentation>
    <partition>
        <name></name>
    </partition>
    <group_data>
        <value></value>
    </group_data>
    <entity_factory>
        <autoenable_created_entities>true</autoenable_created_entities>
    </entity_factory>
</subscriber_qos>
<publisher_qos>
    <presentation>
        <access_scope>INSTANCE_PRESENTATION_QOS</access_scope>
        <coherent_access>false</coherent_access>
        <ordered_access>false</ordered_access>
    </presentation>
    <partition>
        <name></name>
    </partition>
    <group_data>
        <value></value>
    </group_data>
    <entity_factory>
        <autoenable_created_entities>true</autoenable_created_entities>
    </entity_factory>
</publisher_qos>
<topic_qos>
    <topic_data>
        <value></value>
    </topic_data>
    <durability>
        <kind>VOLATILE_DURABILITY_QOS</kind>
    </durability>
    <durability_service>
        <service_cleanup_delay>
            <sec>0</sec>
            <nanosec>0</nanosec>
        </service_cleanup_delay>
        <history_kind>KEEP_LAST_HISTORY_QOS</history_kind>
        <history_depth>1</history_depth>
        <max_samples>LENGTH_UNLIMITED</max_samples>
        <max_instances>LENGTH_UNLIMITED</max_instances>
        <max_samples_per_instance>LENGTH_UNLIMITED</max_samples_per_instance>
    </durability_service>
    <deadline>
        <period>
            <sec>DURATION_INFINITE_SEC</sec>
            <nanosec>DURATION_INFINITE_NSEC</nanosec>
        </period>
```

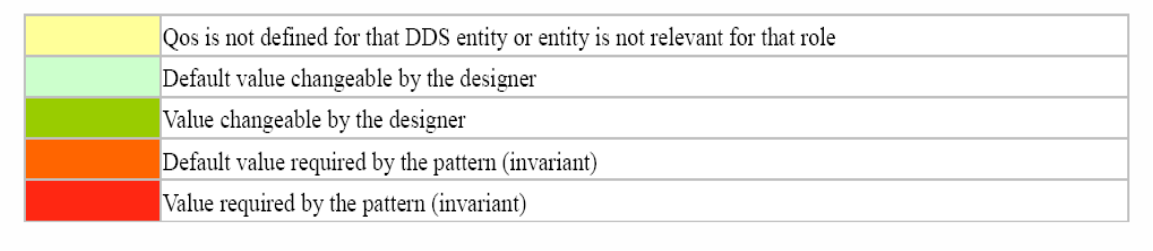
```
            </deadline>
            <latency_budget>
                <duration>
                    <sec>0</sec>
                    <nanosec>0</nanosec>
                </duration>
            </latency_budget>
            <liveliness>
                <kind>AUTOMATIC_LIVELINESS_QOS</kind>
                <lease_duration>
                    <sec>DURATION_INFINITE_SEC</sec>
                    <nanosec>DURATION_INFINITE_NSEC</nanosec>
                </lease_duration>
            </liveliness>
            <reliability>
                <kind>BEST_EFFORT_RELIABILITY_QOS</kind>
                <max_blocking_time>
                    <sec>0</sec>
                    <nanosec>100000000</nanosec>
                    </max_blocking_time>
            </reliability>
            <destination_order>
                <kind>BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS</kind>
            </destination_order>
            <history>
                <kind>KEEP_LAST_HISTORY_QOS</kind>
                <depth>1</depth>
            </history>
            <resource_limits>
                <max_samples>LENGTH_UNLIMITED</max_samples>
                <max_instances>LENGTH_UNLIMITED</max_instances>
                <max_samples_per_instance>LENGTH_UNLIMITED</max_samples_per_instance>
            </resource_limits>
            <transport_priority>
                <value>0</value>
            </transport_priority>
            <lifespan>
                <duration>
                    <sec>DURATION_INFINITE_SEC</sec>
                    <nanosec>DURATION_INFINITE_NSEC</nanosec>
                </duration>
            </lifespan>
            <ownership>
                <kind>SHARED_OWNERSHIP_QOS</kind>
            </ownership>
        </topic_qos>
    </qos_profile>

</dds>
```

Annex E
QoS Policies for the DDS Patterns

(non normative)

The following tables summarizes the DDS QoS policies that are relevant for the two DDS patterns that have been selected (State Transfer Pattern as defined in 7.3.2 ”Pattern State Transfer” on page 20 and Event Transfer Pattern as defined in 7.3.3 ”Pattern Event Transfer ” on page 21).

In those tables the color code is as follows:



🟨	Qos is not defined for that DDS entity or entity is not relevant for that role
🟩	Default value changeable by the designer
🟩	Value changeable by the designer
🟧	Default value required by the pattern (invariant)
🟥	Value required by the pattern (invariant)

Pattern	State				
Role	Observer / State Pattern				
Entity	Topic	Data Reader	Data Writer	Subscriber	Publisher
QoS					
Deadline	infinite	infinite			
Destination order	BY_SOURCE_TIMESTAMP	BY_SOURCE_TIMESTAMP			
Durability	TRANSIENT_LOCAL TRANSIENT	TRANSIENT_LOCAL TRANSIENT			
Durability service					
Entity factory				autoenabled_created_entities=TRUE	
History	KEEP_LAST depth=1	KEEP_LAST depth=1			
Latency budget	0	0			
Lifespan	infinite				
Liveness	AUTOMATIC lease_duration=infinite	AUTOMATIC lease_duration=infinite			
Ownership	SHARED	SHARED			
Partition				""	
Presentation				INSTANCE coherent_access=FALSE ordered_access=TRUE	
Reader data lifecycle		autopurge_nowriter_samples_delay=infinite autopurge_disposed_samples_delay=infinite			
Reliability	RELIABLE	RELIABLE			
Resource limits	max_samples=LENGTH_UNLIMTED max_instances=LENGTH_UNLIMTED max_samples_per_instance=LENGTH_UNLIMTED	max_samples=LENGTH_UNLIMTED max_instances=LENGTH_UNLIMTED max_samples_per_instance=LENGTH_UNLIMTED			
Time based filter		minimum_separation=0			
Transport priority	0				

Pattern	State				
Role	Observable / State Pattern				
Entity	Topic	Data Reader	Data Writer	Subscriber	Publisher
QoS					
Deadline	infinite		infinite		
Destination order	BY_SOURCE_TIMESTAMP		BY_SOURCE_TIMESTAMP		
Durability	TRANSIENT_LOCAL TRANSIENT		TRANSIENT_LOCAL TRANSIENT		
Durability service	service_cleanup_delay=0 history_kind=KEEP_LAST history_depth=1 max_*=LENGTH_UNLIMITED		service_cleanup_delay=0 history_kind=KEEP_LAST history_depth=1 max_*=LENGTH_UNLIMITED		
Entity factory					autoenabled_created_entities=TRUE
History	KEEP_LAST depth=1		KEEP_LAST depth=1		
Latency budget	0		0		
Lifespan	infinite	infinite	infinite		
Liveness	AUTOMATIC lease_duration=infinite	AUTOMATIC lease_duration=infinite	AUTOMATIC lease_duration=infinite		
Ownership	SHARED		SHARED		
Partition					""
Presentation					INSTANCE coherent_access=FALSE ordered_access=TRUE
Reader data lifecycle					
Reliability	RELIABLE		RELIABLE		
Resource limits	max_samples=LENGTH_UNLIMITED max_instances=LENGTH_UNLIMITED max_samples_per_instance=LENGTH_UNLIMITED		max_samples=LENGTH_UNLIMITED max_instances=LENGTH_UNLIMITED max_samples_per_instance=LENGTH_UNLIMITED		
Time based filter					
Transport priority	0		0		

Role	Supplier / Event Pattern				
Entity	Topic	Data Reader	Data Writer	Subscriber	Publisher
Deadline	infinite		infinite		
Destination order	BY_SOURCE_TIMESTAMP		BY_SOURCE_TIMESTAMP		
Durability	VOLATILE		VOLATILE		
Durability service					
Entity factory					autoenabled_created_entities=TRUE
History	KEEP_ALL		KEEP_ALL		
Latency budget	0	0	0		
Lifespan	infinite	infinite	infinite		
Liveness	AUTOMATIC lease_duration=infinite		AUTOMATIC lease_duration=infinite		
Ownership	SHARED		SHARED		
Partition					""
Presentation					INSTANCE coherent_access=FALSE ordered_access=TRUE
Reader data lifecycle					
Reliability	BEST_EFFORT		BEST_EFFORT		
Resource limits	max_samples=LENGTH_UNLIMITED max_instances=LENGTH_UNLIMITED max_samples_per_instance=LENGTH_UNLIMITED	max_samples=LENGTH_UNLIMITED max_instances=LENGTH_UNLIMITED max_samples_per_instance=LENGTH_UNLIMITED	max_samples=LENGTH_UNLIMITED max_instances=LENGTH_UNLIMITED max_samples_per_instance=LENGTH_UNLIMITED		
Time based filter					
Transport priority	0		0		
Writer data lifecycle			autodispose unregistered_instance=FALSE		

Role	Consumer / Event Pattern				
Entity	Topic	Data Reader	Data Writer	Subscriber	Publisher
Deadline	infinite	infinite			
Destination order	BY_SOURCE_TIMESTAMP	BY_SOURCE_TIMESTAMP			
Durability	VOLATILE	VOLATILE			
Durability service					
Entity factory				autoenabled_created_entities=TRUE	
History	KEEP_ALL	KEEP_ALL			
Latency budget	0	0			
Lifespan	infinite				
Liveness	AUTOMATIC lease_duration=infinite	AUTOMATIC lease_duration=infinite			
Ownership	SHARED	SHARED			
Partition				""	
Presentation				INSTANCE coherent_access=FALSE ordered_access=TRUE	
Reader data lifecycle		autopurge_nowriter_samples_delay=infinite autopurge_disposed_samples_delay=infinite			
Reliability	BEST_EFFORT	BEST_EFFORT			
Resource limits	max_samples=LENGTH_UNLIMITED max_instances=LENGTH_UNLIMITED max_samples_per_instance=LENGTH_UNLIMITED	max_samples=LENGTH_UNLIMITED max_instances=LENGTH_UNLIMITED max_samples_per_instance=LENGTH_UNLIMITED			
Time based filter		minimum_separation=0			
Transport priority	0				
Writer data lifecycle					